F:\git\java\mar3\filemonitor\target\contract-module\contract-module-1.doc

0:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\comparisons\IfCmp.java
 */
```
package io.nuls.contract.vm.instructions.comparisons;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class IfCmp {

    public static void ifeq(Frame frame) {
        int value1 = frame.operandStack.popInt();
        int value2 = 0;
        boolean result = value1 == value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "==", value2);
    }

    public static void ifne(Frame frame) {
        int value1 = frame.operandStack.popInt();
        int value2 = 0;
        boolean result = value1 != value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "!=", value2);
    }

    public static void iflt(Frame frame) {
        int value1 = frame.operandStack.popInt();
        int value2 = 0;
        boolean result = value1 < value2;
        if (result) {
            frame.jump();
        }
```

```
        //Log.result(frame.getCurrentOpCode(), result, value1, "<", value2);
    }

    public static void ifge(Frame frame) {
        int value1 = frame.operandStack.popInt();
        int value2 = 0;
        boolean result = value1 >= value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, ">=", value2);
    }

    public static void ifgt(Frame frame) {
        int value1 = frame.operandStack.popInt();
        int value2 = 0;
        boolean result = value1 > value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, ">", value2);
    }

    public static void ifle(Frame frame) {
        int value1 = frame.operandStack.popInt();
        int value2 = 0;
        boolean result = value1 <= value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "<=", value2);
    }

}

1:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\comparisons\IfIcmp.java
 */
package io.nuls.contract.vm.instructions.comparisons;
```

```
import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class IfIcmp {

    public static void if_icmpeq(Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        boolean result = value1 == value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "==", value2);
    }

    public static void if_icmpne(Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        boolean result = value1 != value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "!=", value2);
    }

    public static void if_icmplt(Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        boolean result = value1 < value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "<", value2);
    }

    public static void if_icmpge(Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
```

```
        boolean result = value1 >= value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, ">=", value2);
    }

    public static void if_icmpgt(Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        boolean result = value1 > value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, ">", value2);
    }

    public static void if_icmple(Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        boolean result = value1 <= value2;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value1, "<=", value2);
    }

}

2:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\comparisons\Lcmp.java
 */
package io.nuls.contract.vm.instructions.comparisons;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Lcmp {
```

```
    public static void lcmp(Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        int result = Long.compare(value1, value2);
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "compare", value2);
    }

}

3:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\constants\Aconst.java
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Aconst {

    public static void aconst_null(final Frame frame) {
        frame.operandStack.pushRef(null);

        //Log.opcode(frame.getCurrentOpCode());
    }

}

4:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\constants\Dconst.java
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Dconst {

    public static void dconst_0(Frame frame) {
        dconst(frame, 0.0D);
    }
```

```
    public static void dconst_1(Frame frame) {
        dconst(frame, 1.0D);
    }

    private static void dconst(Frame frame, double value) {
        frame.operandStack.pushDouble(value);

        //Log.opcode(frame.getCurrentOpCode());
    }

}

5:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\constants\Fconst.java
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Fconst {

    public static void fconst_0(final Frame frame) {
        fconst(frame, 0.0F);
    }

    public static void fconst_1(final Frame frame) {
        fconst(frame, 1.0F);
    }

    public static void fconst_2(final Frame frame) {
        fconst(frame, 2.0F);
    }

    private static void fconst(Frame frame, float value) {
        frame.operandStack.pushFloat(value);

        //Log.opcode(frame.getCurrentOpCode());
    }

}
```

```
6:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\constants\Iconst.java
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Iconst {

    public static void iconst_m1(final Frame frame) {
        iconst(frame, -1);
    }

    public static void iconst_0(final Frame frame) {
        iconst(frame, 0);
    }

    public static void iconst_1(final Frame frame) {
        iconst(frame, 1);
    }

    public static void iconst_2(final Frame frame) {
        iconst(frame, 2);
    }

    public static void iconst_3(final Frame frame) {
        iconst(frame, 3);
    }

    public static void iconst_4(final Frame frame) {
        iconst(frame, 4);
    }

    public static void iconst_5(final Frame frame) {
        iconst(frame, 5);
    }

    private static void iconst(Frame frame, int value) {
        frame.operandStack.pushInt(value);
```

```
        //Log.opcode(frame.getCurrentOpCode());
    }

}
```

7:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\constants\Lconst.java

```
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Lconst {

    public static void lconst_0(final Frame frame) {
        lconst(frame, 0L);
    }

    public static void lconst_1(final Frame frame) {
        lconst(frame, 1L);
    }

    private static void lconst(Frame frame, long value) {
        frame.operandStack.pushLong(value);

        //Log.opcode(frame.getCurrentOpCode());
    }

}
```

8:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\constants\Ldc.java

```
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.Type;

public class Ldc {
```

```
    public static void ldc(final Frame frame) {
        Object value = frame.ldcInsnNode().cst;

        //Log.opcode(frame.getCurrentOpCode(), value);

        if (value instanceof Integer) {
            frame.operandStack.pushInt((int) value);
        } else if (value instanceof Long) {
            frame.operandStack.pushLong((long) value);
        } else if (value instanceof Float) {
            frame.operandStack.pushFloat((float) value);
        } else if (value instanceof Double) {
            frame.operandStack.pushDouble((double) value);
        } else if (value instanceof String) {
            String str = (String) value;
            ObjectRef objectRef = frame.heap.newString(str);
            frame.operandStack.pushRef(objectRef);
        } else if (value instanceof Type) {
            Type type = (Type) value;
            String desc = type.getDescriptor();
            ObjectRef objectRef = frame.heap.getClassRef(desc);
            frame.operandStack.pushRef(objectRef);
        } else {
            throw new IllegalArgumentException("unknown ldc cst");
        }
    }

}

9:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\constants\Nop.java
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Nop {

    public static void nop(Frame frame) {
        //Log.opcode(frame.getCurrentOpCode());
```

```
    }

}
```

10:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\constants\Xipush.java

```
 */
package io.nuls.contract.vm.instructions.constants;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Xipush {

    public static void bipush(final Frame frame) {
        xipush(frame);
    }

    public static void sipush(final Frame frame) {
        xipush(frame);
    }

    private static void xipush(final Frame frame) {
        int value = frame.intInsnNode().operand;
        frame.operandStack.pushInt(value);

        //Log.opcode(frame.getCurrentOpCode(), value);
    }

}
```

11:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\control\Goto.java

```
 */
package io.nuls.contract.vm.instructions.control;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Goto {

    public static void goto_(final Frame frame) {
```

```
        frame.jump();

        //Log.opcode(frame.getCurrentOpCode());
    }

}
```

12:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\control\Jsr.java

```
 */
package io.nuls.contract.vm.instructions.control;

import io.nuls.contract.vm.Frame;

public class Jsr {

    public static void jsr(final Frame frame) {
        frame.nonsupportOpCode();
    }

}
```

13:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\control\Lookupswitch.java

```
 */
package io.nuls.contract.vm.instructions.control;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.LabelNode;
import org.objectweb.asm.tree.LookupSwitchInsnNode;

public class Lookupswitch {

    public static void lookupswitch(final Frame frame) {
        LookupSwitchInsnNode lookup = frame.lookupSwitchInsnNode();
        LabelNode labelNode = lookup.dflt;
        int key = frame.operandStack.popInt();
        for (int i = 0; i < lookup.keys.size(); i++) {
            int k = lookup.keys.get(i);
            if (k == key) {
                labelNode = lookup.labels.get(i);
```

```
                break;
            }
        }
        frame.jump(labelNode);

        //Log.opcode(frame.getCurrentOpCode());
    }

}
```

14:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\control\Ret.java

```
 */
package io.nuls.contract.vm.instructions.control;

import io.nuls.contract.vm.Frame;

public class Ret {

    public static void ret(final Frame frame) {
        frame.nonsupportOpCode();
    }

}
```

15:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\control\Return.java

```
 */
package io.nuls.contract.vm.instructions.control;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.util.Log;

public class Return {

    public static void ireturn(final Frame frame) {
        Object result;
        switch (frame.result.getVariableType().getType()) {
            case Descriptors.BOOLEAN:
                result = frame.operandStack.popBoolean();
```

```
            break;
        case Descriptors.BYTE:
            result = frame.operandStack.popByte();
            break;
        case Descriptors.CHAR:
            result = frame.operandStack.popChar();
            break;
        case Descriptors.SHORT:
            result = frame.operandStack.popShort();
            break;
        default:
            result = frame.operandStack.popInt();
            break;
    }
    frame.result.value(result);

    //Log.result(frame.getCurrentOpCode(), result);
}

public static void lreturn(final Frame frame) {
    long result = frame.operandStack.popLong();
    frame.result.value(result);

    //Log.result(frame.getCurrentOpCode(), result);
}

public static void freturn(final Frame frame) {
    float result = frame.operandStack.popFloat();
    frame.result.value(result);

    //Log.result(frame.getCurrentOpCode(), result);
}

public static void dreturn(final Frame frame) {
    double result = frame.operandStack.popDouble();
    frame.result.value(result);

    //Log.result(frame.getCurrentOpCode(), result);
}

public static void areturn(final Frame frame) {
    ObjectRef result = frame.operandStack.popRef();
```

```
        frame.result.value(result);

        //Log.result(frame.getCurrentOpCode(), result);
    }

    public static void return_(final Frame frame) {
        frame.result.value(null);

        //Log.opcode(frame.getCurrentOpCode());
    }

}

16:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\control\Tableswitch.java
 */
package io.nuls.contract.vm.instructions.control;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.LabelNode;
import org.objectweb.asm.tree.TableSwitchInsnNode;

public class Tableswitch {

    public static void tableswitch(final Frame frame) {
        TableSwitchInsnNode table = frame.tableSwitchInsnNode();
        LabelNode labelNode = table.dflt;
        int index = frame.operandStack.popInt();
        int min = table.min;
        int max = table.max;
        int size = max - min + 1;
        for (int i = 0; i < size; i++) {
            if (index == (i + min)) {
                labelNode = table.labels.get(i);
                break;
            }
        }
        frame.jump(labelNode);

        //Log.opcode(frame.getCurrentOpCode());
    }
```

```
}

17:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\conversions\D2x.java
 */
package io.nuls.contract.vm.instructions.conversions;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class D2x {

    public static void d2i(Frame frame) {
        double value = frame.operandStack.popDouble();
        int result = (int) value;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

    public static void d2l(Frame frame) {
        double value = frame.operandStack.popDouble();
        long result = (long) value;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

    public static void d2f(Frame frame) {
        double value = frame.operandStack.popDouble();
        float result = (float) value;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

}

18:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\conversions\F2x.java
 */
```

```
package io.nuls.contract.vm.instructions.conversions;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class F2x {

    public static void f2i(Frame frame) {
        float value = frame.operandStack.popFloat();
        int result = (int) value;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

    public static void f2l(Frame frame) {
        float value = frame.operandStack.popFloat();
        long result = (long) value;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

    public static void f2d(Frame frame) {
        float value = frame.operandStack.popFloat();
        double result = (double) value;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

}

19:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\conversions\I2x.java
 */
package io.nuls.contract.vm.instructions.conversions;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class I2x {
```

```
public static void i2l(Frame frame) {
    int value = frame.operandStack.popInt();
    long result = (long) value;
    frame.operandStack.pushLong(result);

    //Log.result(frame.getCurrentOpCode(), result, value);
}

public static void i2f(Frame frame) {
    int value = frame.operandStack.popInt();
    float result = (float) value;
    frame.operandStack.pushFloat(result);

    //Log.result(frame.getCurrentOpCode(), result, value);
}

public static void i2d(Frame frame) {
    int value = frame.operandStack.popInt();
    double result = (double) value;
    frame.operandStack.pushDouble(result);

    //Log.result(frame.getCurrentOpCode(), result, value);
}

public static void i2b(Frame frame) {
    int value = frame.operandStack.popInt();
    byte result = (byte) value;
    frame.operandStack.pushByte(result);

    //Log.result(frame.getCurrentOpCode(), result, value);
}

public static void i2c(Frame frame) {
    int value = frame.operandStack.popInt();
    char result = (char) value;
    frame.operandStack.pushChar(result);

    //Log.result(frame.getCurrentOpCode(), result, value);
}

public static void i2s(Frame frame) {
```

```
        int value = frame.operandStack.popInt();
        short result = (short) value;
        frame.operandStack.pushShort(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

}

20:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\conversions\L2x.java
 */
package io.nuls.contract.vm.instructions.conversions;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class L2x {

    public static void l2i(Frame frame) {
        long value = frame.operandStack.popLong();
        int result = (int) value;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

    public static void l2f(Frame frame) {
        long value = frame.operandStack.popLong();
        float result = (float) value;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }

    public static void l2d(Frame frame) {
        long value = frame.operandStack.popLong();
        double result = (double) value;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value);
    }
```

```
}

21:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\extended\Ifnonnull.java
 */
package io.nuls.contract.vm.instructions.extended;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class Ifnonnull {

    public static void ifnonnull(final Frame frame) {
        ObjectRef value = frame.operandStack.popRef();
        boolean result = value != null;
        if (result) {
            frame.jump();
        }

        //Log.result(frame.getCurrentOpCode(), result, value, "!=", null);
    }

}

22:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\extended\Ifnull.java
 */
package io.nuls.contract.vm.instructions.extended;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class Ifnull {

    public static void ifnull(final Frame frame) {
        ObjectRef value = frame.operandStack.popRef();
        boolean result = value == null;
        if (result) {
            frame.jump();
```

```
        }

        //Log.result(frame.getCurrentOpCode(), result, value, "==", null);
    }

}
```

23:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\extended\Multianewarray.java

```
 */
package io.nuls.contract.vm.instructions.extended;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.MultiANewArrayInsnNode;

public class Multianewarray {

    public static void multianewarray(final Frame frame) {
        MultiANewArrayInsnNode multiANewArrayInsnNode = frame.multiANewArrayInsnNode();
        int[] dimensions = new int[multiANewArrayInsnNode.dims];
        for (int i = multiANewArrayInsnNode.dims - 1; i >= 0; i--) {
            int length = frame.operandStack.popInt();
            if (length < 0) {
                frame.throwNegativeArraySizeException();
                return;
            }
            dimensions[i] = length;
        }
        VariableType variableType = VariableType.valueOf(multiANewArrayInsnNode.desc);
        ObjectRef arrayRef = frame.heap.newArray(variableType, dimensions);
        frame.operandStack.pushRef(arrayRef);

        //Log.result(frame.getCurrentOpCode(), arrayRef);
    }

}
```

24:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-

vm\src\main\java\io\nuls\contract\vm\instructions\loads\Aload.java
 */
```
package io.nuls.contract.vm.instructions.loads;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class Aload {

    public static void aload(final Frame frame) {
        int index = frame.varInsnNode().var;
        ObjectRef objectRef = frame.localVariables.getRef(index);
        frame.operandStack.pushRef(objectRef);

        //Log.result(frame.getCurrentOpCode(), objectRef, index);
    }

}
```

25:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\loads\Dload.java
 */
```
package io.nuls.contract.vm.instructions.loads;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Dload {

    public static void dload(final Frame frame) {
        int index = frame.varInsnNode().var;
        double value = frame.localVariables.getDouble(index);
        frame.operandStack.pushDouble(value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

26:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\loads\Fload.java

```
 */
package io.nuls.contract.vm.instructions.loads;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Fload {

    public static void fload(final Frame frame) {
        int index = frame.varInsnNode().var;
        float value = frame.localVariables.getFloat(index);
        frame.operandStack.pushFloat(value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

27:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\loads\Iload.java

```
 */
package io.nuls.contract.vm.instructions.loads;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Iload {

    public static void iload(final Frame frame) {
        int index = frame.varInsnNode().var;
        int value = frame.localVariables.getInt(index);
        frame.operandStack.pushInt(value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

28:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\loads\Lload.java

```
 */
package io.nuls.contract.vm.instructions.loads;
```

```
import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Lload {

    public static void lload(final Frame frame) {
        int index = frame.varInsnNode().var;
        long value = frame.localVariables.getLong(index);
        frame.operandStack.pushLong(value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}

29:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\loads\Xaload.java
 */
package io.nuls.contract.vm.instructions.loads;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class Xaload {

    public static void iaload(final Frame frame) {
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        int value = (int) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushInt(value);

        //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
    }

    public static void laload(final Frame frame) {
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
```

```
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        long value = (long) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushLong(value);

        //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
    }

    public static void faload(final Frame frame) {
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        float value = (float) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushFloat(value);

        //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
    }

    public static void daload(final Frame frame) {
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        double value = (double) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushDouble(value);

        //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
    }

    public static void aaload(final Frame frame) {
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        ObjectRef value = (ObjectRef) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushRef(value);
```

```
    //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
}

public static void baload(final Frame frame) {
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
    }
    Object result;
    if (arrayRef.getVariableType().getComponentType().isBoolean()) {
        boolean value = (boolean) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushBoolean(value);
        result = value;
    } else {
        byte value = (byte) frame.heap.getArray(arrayRef, index);
        frame.operandStack.pushByte(value);
        result = value;
    }

    //Log.result(frame.getCurrentOpCode(), result, arrayRef, index);
}

public static void caload(final Frame frame) {
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
    }
    char value = (char) frame.heap.getArray(arrayRef, index);
    frame.operandStack.pushChar(value);

    //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
}

public static void saload(final Frame frame) {
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
    }
    short value = (short) frame.heap.getArray(arrayRef, index);
```

```
        frame.operandStack.pushShort(value);

        //Log.result(frame.getCurrentOpCode(), value, arrayRef, index);
    }

}

30:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Add.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Add {

    public static void iadd(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 + value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "+", value2);
    }

    public static void ladd(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 + value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "+", value2);
    }

    public static void fadd(final Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        float result = value1 + value2;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "+", value2);
```

```
    }

    public static void dadd(final Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        double result = value1 + value2;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "+", value2);
    }

}

31:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\And.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class And {

    public static void iand(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 & value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "&", value2);
    }

    public static void land(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 & value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "&", value2);
    }

}
```

```
32:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Div.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Div {

    public static void idiv(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 / value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "/", value2);
    }

    public static void ldiv(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 / value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "/", value2);
    }

    public static void fdiv(final Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        float result = value1 / value2;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "/", value2);
    }

    public static void ddiv(final Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        double result = value1 / value2;
```

```
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "/", value2);
    }

}
```

33:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\math\Iinc.java

```
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.IincInsnNode;

public class Iinc {

    public static void iinc(final Frame frame) {
        IincInsnNode iincInsnNode = frame.iincInsnNode();
        int index = iincInsnNode.var;
        int incr = iincInsnNode.incr;
        int value = frame.localVariables.getInt(index);
        int result = value + incr;
        frame.localVariables.setInt(index, result);

        //Log.result(frame.getCurrentOpCode(), result, value, "+", incr);
    }

}
```

34:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\math\Mul.java

```
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Mul {

    public static void imul(final Frame frame) {
```

```
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 * value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "*", value2);
    }

    public static void lmul(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 * value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "*", value2);
    }

    public static void fmul(final Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        float result = value1 * value2;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "*", value2);
    }

    public static void dmul(final Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        double result = value1 * value2;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "*", value2);
    }

}

35:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Neg.java
 */
package io.nuls.contract.vm.instructions.math;
```

```
import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Neg {

    public static void ineg(final Frame frame) {
        int value = frame.operandStack.popInt();
        int result = -value;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, "-", value);
    }

    public static void lneg(final Frame frame) {
        long value = frame.operandStack.popLong();
        long result = -value;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, "-", value);
    }

    public static void fneg(final Frame frame) {
        float value = frame.operandStack.popFloat();
        float result = -value;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, "-", value);
    }

    public static void dneg(final Frame frame) {
        double value = frame.operandStack.popDouble();
        double result = -value;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, "-", value);
    }

}

36:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Or.java
 */
```

```
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Or {

    public static void ior(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 | value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "|", value2);
    }

    public static void lor(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 | value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "|", value2);
    }

}

37:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Rem.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Rem {

    public static void irem(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 % value2;
        frame.operandStack.pushInt(result);
```

```
        //Log.result(frame.getCurrentOpCode(), result, value1, "%", value2);
    }

    public static void lrem(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 % value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "%", value2);
    }

    public static void frem(final Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        float result = value1 % value2;
        frame.operandStack.pushFloat(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "%", value2);
    }

    public static void drem(final Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        double result = value1 % value2;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "%", value2);
    }

}

38:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Shl.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Shl {
```

```
    public static void ishl(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 << value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "<<", value2);
    }

    public static void lshl(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        long value1 = frame.operandStack.popLong();
        long result = value1 << value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "<<", value2);
    }

}

39:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Shr.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Shr {

    public static void ishr(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 >> value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, ">>", value2);
    }

    public static void lshr(final Frame frame) {
        int value2 = frame.operandStack.popInt();
```

```
        long value1 = frame.operandStack.popLong();
        long result = value1 >> value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, ">>", value2);
    }

}

40:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Sub.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Sub {

    public static void isub(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 - value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "-", value2);
    }

    public static void lsub(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 - value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "-", value2);
    }

    public static void fsub(final Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        float result = value1 - value2;
        frame.operandStack.pushFloat(result);
```

```
        //Log.result(frame.getCurrentOpCode(), result, value1, "-", value2);
    }

    public static void dsub(final Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        double result = value1 - value2;
        frame.operandStack.pushDouble(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "-", value2);
    }

}

41:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Ushr.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Ushr {

    public static void iushr(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 >>> value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, ">>>", value2);
    }

    public static void lushr(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        long value1 = frame.operandStack.popLong();
        long result = value1 >>> value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, ">>>", value2);
    }
```

```
}

42:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\math\Xor.java
 */
package io.nuls.contract.vm.instructions.math;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Xor {

    public static void ixor(final Frame frame) {
        int value2 = frame.operandStack.popInt();
        int value1 = frame.operandStack.popInt();
        int result = value1 ^ value2;
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "^", value2);
    }

    public static void lxor(final Frame frame) {
        long value2 = frame.operandStack.popLong();
        long value1 = frame.operandStack.popLong();
        long result = value1 ^ value2;
        frame.operandStack.pushLong(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "^", value2);
    }

}

43:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Anewarray.java
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.util.Constants;
```

```
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.TypeInsnNode;

public class Anewarray {

    public static void anewarray(Frame frame) {
        TypeInsnNode typeInsnNode = frame.typeInsnNode();
        String className = typeInsnNode.desc;
        int length = frame.operandStack.popInt();
        if (length < 0) {
            frame.throwNegativeArraySizeException();
            return;
        } else {
            ObjectRef arrayRef;
            if (className.contains(Constants.ARRAY_START)) {
                VariableType type = VariableType.valueOf(className);
                int[] dimensions = new int[type.getDimensions() + 1];
                dimensions[0] = length;
                VariableType variableType = VariableType.valueOf(Constants.ARRAY_START +
className);
                arrayRef = frame.heap.newArray(variableType, dimensions);
            } else {
                VariableType variableType = VariableType.valueOf(Constants.ARRAY_PREFIX +
className + Constants.ARRAY_SUFFIX);
                arrayRef = frame.heap.newArray(variableType, length);
            }
            frame.operandStack.pushRef(arrayRef);

            //Log.opcode(frame.getCurrentOpCode(), arrayRef);
        }
    }

}

44:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Arraylength.java
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;
```

```
public class Arraylength {

    public static void arraylength(Frame frame) {
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (arrayRef == null) {
            frame.throwNullPointerException();
            return;
        }
        int length = arrayRef.getDimensions()[0];
        frame.operandStack.pushInt(length);

        //Log.result(frame.getCurrentOpCode(), length);
    }

}

45:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Athrow.java
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.util.Constants;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.TryCatchBlockNode;

import java.util.Objects;

public class Athrow {

    public static void athrow(final Frame frame) {
        ObjectRef objectRef = frame.operandStack.popRef();
        if (objectRef == null) {
            frame.throwNullPointerException();
            return;
        }
        //Log.opcode(frame.getCurrentOpCode(), objectRef);
        while (frame.vm.isNotEmptyFrame()) {
            final Frame lastFrame = frame.vm.lastFrame();
```

```
            TryCatchBlockNode tryCatchBlockNode = getTryCatchBlockNode(lastFrame, objectRef);
            if (tryCatchBlockNode != null) {
                lastFrame.operandStack.clear();
                lastFrame.operandStack.pushRef(objectRef);
                lastFrame.jump(tryCatchBlockNode.handler);
                return;
            } else {
                frame.vm.popFrame();
            }
        }
        frame.vm.getResult().exception(objectRef);
    }

    private static TryCatchBlockNode getTryCatchBlockNode(Frame frame, ObjectRef objectRef) {
        for (TryCatchBlockNode tryCatchBlockNode : frame.methodCode.tryCatchBlocks) {
            String type = tryCatchBlockNode.type;
            int line = frame.getLine();
            int start = frame.getLine(tryCatchBlockNode.start);
            int end = frame.getLine(tryCatchBlockNode.end);
            int handler = frame.getLine(tryCatchBlockNode.handler);
            if (type != null && handler < end) {
                end = handler;
            }
            boolean result = start <= line && line < end;
            if (result && (type == null || extends_(objectRef.getVariableType().getType(), type, frame)))
{
                return tryCatchBlockNode;
            }
        }
        return null;
    }

    private static boolean extends_(String refType, String className, Frame frame) {
        if (Objects.equals(refType, className)) {
            return true;
        } else {
            ClassCode classCode = frame.methodArea.loadClass(refType);
            String superName = classCode.superName;
            if (Constants.OBJECT_CLASS_NAME.equals(superName)) {
                return false;
            } else {
                return extends_(superName, className, frame);
```

```
        }
      }
   }

}
```

46:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Checkcast.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.TypeInsnNode;

public class Checkcast {

    public static void checkcast(Frame frame) {
        TypeInsnNode typeInsnNode = frame.typeInsnNode();
        String desc = typeInsnNode.desc;
        VariableType variableType = VariableType.valueOf(desc);
        ObjectRef objectRef = frame.operandStack.popRef();

        if (objectRef == null || Instanceof.instanceof_(objectRef, variableType, frame)) {
            frame.operandStack.pushRef(objectRef);
        } else {
            frame.throwClassCastException();
        }

        //Log.opcode(frame.getCurrentOpCode(), objectRef, desc);
    }

}
```

47:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Getfield.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
```

```
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.FieldInsnNode;

public class Getfield {

    public static void getfield(Frame frame) {
        FieldInsnNode fieldInsnNode = frame.fieldInsnNode();
        String fieldName = fieldInsnNode.name;
        String fieldDesc = fieldInsnNode.desc;
        ObjectRef objectRef = frame.operandStack.popRef();
        if (objectRef == null) {
            frame.throwNullPointerException();
            return;
        }
        Object value = frame.heap.getField(objectRef, fieldName);
        if (Descriptors.LONG_DESC.equals(fieldDesc)) {
            frame.operandStack.pushLong((long) value);
        } else if (Descriptors.DOUBLE_DESC.equals(fieldDesc)) {
            frame.operandStack.pushDouble((double) value);
        } else {
            frame.operandStack.push(value);
        }

        //Log.result(frame.getCurrentOpCode(), value, objectRef, fieldName);
    }

}

48:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Getstatic.java
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.FieldInsnNode;

public class Getstatic {
```

```
    public static void getstatic(Frame frame) {
        FieldInsnNode fieldInsnNode = frame.fieldInsnNode();
        String className = fieldInsnNode.owner;
        String fieldName = fieldInsnNode.name;
        String fieldDesc = fieldInsnNode.desc;
        Object value = frame.heap.getStatic(className, fieldName);
        if (Descriptors.LONG_DESC.equals(fieldDesc)) {
            frame.operandStack.pushLong((long) value);
        } else if (Descriptors.DOUBLE_DESC.equals(fieldDesc)) {
            frame.operandStack.pushDouble((double) value);
        } else {
            frame.operandStack.push(value);
        }

        //Log.result(frame.getCurrentOpCode(), value, className, fieldName);
    }

}
```

49:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Instanceof.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.util.Constants;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.TypeInsnNode;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Instanceof {

    public static void instanceof_(Frame frame) {
        TypeInsnNode typeInsnNode = frame.typeInsnNode();
        VariableType variableType = VariableType.valueOf(typeInsnNode.desc);
        ObjectRef objectRef = frame.operandStack.popRef();
```

```
        boolean result = instanceof_(objectRef, variableType, frame);
        frame.operandStack.pushInt(result ? 1 : 0);

        //Log.result(frame.getCurrentOpCode(), result, objectRef, variableType);
    }

    public static boolean instanceof_(ObjectRef objectRef, VariableType variableType, Frame
frame) {
        boolean result;
        if (objectRef == null) {
            result = false;
        } else if (Objects.equals(Constants.OBJECT_CLASS_DESC, variableType.getDesc())) {
            result = true;
        } else if (objectRef.isArray() || variableType.isArray()) {
            if (objectRef.isArray() && variableType.isArray()) {
                if (objectRef.getDimensions().length == variableType.getDimensions()) {
                    result = instanceof_(objectRef.getVariableType().getType(), variableType.getType(),
frame);
                } else {
                    result = false;
                }
            } else {
                result = false;
            }
        } else {
            result = instanceof_(objectRef.getVariableType().getType(), variableType.getType(),
frame);
        }
        return result;
    }

    public static boolean instanceof_(String refType, String className, Frame frame) {
        if (Objects.equals(refType, className) ||
Objects.equals(Constants.OBJECT_CLASS_NAME, className)) {
            return true;
        } else if (Objects.equals(Constants.OBJECT_CLASS_NAME, refType)) {
            return false;
        } else {
            ClassCode classCode = frame.methodArea.loadClass(refType);
            String superName = classCode.superName;
            List<String> list = new ArrayList<>();
            list.add(superName);
```

```
            list.addAll(classCode.interfaces);

            for (String s : list) {
                boolean result = instanceof_(s, className, frame);
                if (result) {
                    return true;
                }
            }
            return false;
        }
    }

}
```

50:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Invokedynamic.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;

public class Invokedynamic {

    public static void invokedynamic(Frame frame) {
        frame.nonsupportOpCode();
    }

}
```

51:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Invokeinterface.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.MethodInsnNode;
```

```
import java.util.List;

public class Invokeinterface {

    public static void invokeinterface(Frame frame) {
        MethodInsnNode methodInsnNode = frame.methodInsnNode();
        String interfaceName = methodInsnNode.owner;
        String interfaceMethodName = methodInsnNode.name;
        String interfaceMethodDesc = methodInsnNode.desc;

        List<VariableType> variableTypes = VariableType.parseArgs(interfaceMethodDesc);
        MethodArgs methodArgs = new MethodArgs(variableTypes, frame.operandStack, false);
        ObjectRef objectRef = methodArgs.objectRef;
        if (objectRef == null) {
            frame.throwNullPointerException();
            return;
        }

        String className = objectRef.getVariableType().getType();
        MethodCode methodCode = frame.methodArea.loadMethod(className,
interfaceMethodName, interfaceMethodDesc);

        //Log.opcode(frame.getCurrentOpCode(), className, interfaceMethodName,
interfaceMethodDesc);

        frame.vm.run(methodCode, methodArgs.frameArgs, true);
    }

}

52:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Invokespecial.java
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;
import io.nuls.contract.vm.util.Constants;
```

```
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.MethodInsnNode;

public class Invokespecial {

    public static void invokespecial(Frame frame) {
        MethodInsnNode methodInsnNode = frame.methodInsnNode();
        String className = methodInsnNode.owner;
        String methodName = methodInsnNode.name;
        String methodDesc = methodInsnNode.desc;

        MethodCode methodCode = frame.methodArea.loadMethod(className, methodName,
methodDesc);

        MethodArgs methodArgs = new MethodArgs(methodCode.argsVariableType,
frame.operandStack, false);
        ObjectRef objectRef = methodArgs.objectRef;
        if (objectRef == null) {
            frame.throwNullPointerException();
            return;
        }

        //Log.opcode(frame.getCurrentOpCode(), className, methodName, methodDesc);

        if (Constants.OBJECT_CLASS_NAME.equals(className) &&
Constants.CONSTRUCTOR_NAME.equals(methodName)) {
            return;
        }

        Result result = NativeMethod.run(methodCode, methodArgs, frame);
        if (result != null) {
            return;
        }

        frame.vm.run(methodCode, methodArgs.frameArgs, true);
    }

}

53:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Invokestatic.java
 */
```

```
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.MethodInsnNode;

public class Invokestatic {

    public static void invokestatic(Frame frame) {
        MethodInsnNode methodInsnNode = frame.methodInsnNode();
        String className = methodInsnNode.owner;
        String methodName = methodInsnNode.name;
        String methodDesc = methodInsnNode.desc;

        MethodCode methodCode = frame.methodArea.loadMethod(className, methodName,
methodDesc);

        MethodArgs methodArgs = new MethodArgs(methodCode.argsVariableType,
frame.operandStack, true);

        //Log.opcode(frame.getCurrentOpCode(), className, methodName, methodDesc);

        Result result = NativeMethod.run(methodCode, methodArgs, frame);
        if (result != null) {
            return;
        }

        frame.vm.run(methodCode, methodArgs.frameArgs, true);
    }

}

54:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\references\Invokevirtual.java
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
```

```
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.natives.NativeMethod;
import io.nuls.contract.vm.util.Constants;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.MethodInsnNode;

import java.util.List;
import java.util.Objects;

public class Invokevirtual {

    public static void invokevirtual(Frame frame) {
        MethodInsnNode methodInsnNode = frame.methodInsnNode();
        String className = methodInsnNode.owner;
        String methodName = methodInsnNode.name;
        String methodDesc = methodInsnNode.desc;

        List<VariableType> variableTypes = VariableType.parseArgs(methodDesc);
        MethodArgs methodArgs = new MethodArgs(variableTypes, frame.operandStack, false);
        ObjectRef objectRef = methodArgs.objectRef;
        if (objectRef == null) {
            frame.throwNullPointerException();
            return;
        }

        String type = objectRef.getVariableType().getType();

        if (!Objects.equals(className, type)) {
            if (objectRef.getVariableType().isPrimitiveType()) {

            } else {
                className = type;
            }
        }

        if (objectRef.isArray() && Constants.TO_STRING_METHOD_NAME.equals(methodName)
&& Constants.TO_STRING_METHOD_DESC.equals(methodDesc)) {
            className = Constants.OBJECT_CLASS_NAME;
```

```
        }

        MethodCode methodCode = frame.methodArea.loadMethod(className, methodName,
methodDesc);

        //Log.opcode(frame.getCurrentOpCode(), objectRef, methodName, methodDesc);

        Result result = NativeMethod.run(methodCode, methodArgs, frame);
        if (result != null) {
            return;
        }

        frame.vm.run(methodCode, methodArgs.frameArgs, true);
    }

}
```

55:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Monitorenter.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;

public class Monitorenter {

    public static void monitorenter(Frame frame) {
        ObjectRef objectRef = frame.operandStack.popRef();
        //frame.nonsupportOpCode();
    }

}
```

56:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Monitorexit.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
```

```
public class Monitorexit {

    public static void monitorexit(Frame frame) {
        ObjectRef objectRef = frame.operandStack.popRef();
        //frame.nonsupportOpCode();
    }

}
```

57:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\New.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class New {

    public static void new_(Frame frame) {
        String className = frame.typeInsnNode().desc;
        ObjectRef objectRef = frame.heap.newObject(className);
        frame.operandStack.pushRef(objectRef);

        //Log.opcode(frame.getCurrentOpCode(), objectRef);
    }

}
```

58:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Newarray.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.Opcodes;

public class Newarray {
```

```
public static void newarray(Frame frame) {
    VariableType type;
    int length = frame.operandStack.popInt();
    if (length < 0) {
        frame.throwNegativeArraySizeException();
        return;
    } else {
        switch (frame.intInsnNode().operand) {
            case Opcodes.T_BOOLEAN:
                type = VariableType.BOOLEAN_ARRAY_TYPE;
                break;
            case Opcodes.T_CHAR:
                type = VariableType.CHAR_ARRAY_TYPE;
                break;
            case Opcodes.T_FLOAT:
                type = VariableType.FLOAT_ARRAY_TYPE;
                break;
            case Opcodes.T_DOUBLE:
                type = VariableType.DOUBLE_ARRAY_TYPE;
                break;
            case Opcodes.T_BYTE:
                type = VariableType.BYTE_ARRAY_TYPE;
                break;
            case Opcodes.T_SHORT:
                type = VariableType.SHORT_ARRAY_TYPE;
                break;
            case Opcodes.T_INT:
                type = VariableType.INT_ARRAY_TYPE;
                break;
            case Opcodes.T_LONG:
                type = VariableType.LONG_ARRAY_TYPE;
                break;
            default:
                throw new IllegalArgumentException("unknown operand");
        }
    }

    ObjectRef arrayRef = frame.heap.newArray(type, length);
    frame.operandStack.pushRef(arrayRef);

    //Log.result(frame.getCurrentOpCode(), arrayRef);
```

```
    }

}
```

59:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\references\Putfield.java

```
 */
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.FieldInsnNode;

public class Putfield {

    public static void putfield(Frame frame) {
        FieldInsnNode fieldInsnNode = frame.fieldInsnNode();
        String fieldName = fieldInsnNode.name;
        String fieldDesc = fieldInsnNode.desc;
        Object value;
        if (Descriptors.LONG_DESC.equals(fieldDesc)) {
            value = frame.operandStack.popLong();
        } else if (Descriptors.DOUBLE_DESC.equals(fieldDesc)) {
            value = frame.operandStack.popDouble();
        } else {
            value = frame.operandStack.pop();
        }
        ObjectRef objectRef = frame.operandStack.popRef();
        if (objectRef == null) {
            frame.throwNullPointerException();
            return;
        }
        frame.heap.putField(objectRef, fieldName, value);

        //Log.result(frame.getCurrentOpCode(), value, objectRef, fieldName);
    }

}
```

60:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-

vm\src\main\java\io\nuls\contract\vm\instructions\references\Putstatic.java
 */
```
package io.nuls.contract.vm.instructions.references;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.util.Log;
import org.objectweb.asm.tree.FieldInsnNode;

public class Putstatic {

    public static void putstatic(Frame frame) {
        FieldInsnNode fieldInsnNode = frame.fieldInsnNode();
        String className = fieldInsnNode.owner;
        String fieldName = fieldInsnNode.name;
        String fieldDesc = fieldInsnNode.desc;
        Object value;
        if (Descriptors.LONG_DESC.equals(fieldDesc)) {
            value = frame.operandStack.popLong();
        } else if (Descriptors.DOUBLE_DESC.equals(fieldDesc)) {
            value = frame.operandStack.popDouble();
        } else {
            value = frame.operandStack.pop();
        }
        frame.heap.putStatic(className, fieldName, value);

        //Log.result(frame.getCurrentOpCode(), value, className, fieldName);
    }

}
```

61:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stack\Dup.java
 */
```
package io.nuls.contract.vm.instructions.stack;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Dup {

    public static void dup(final Frame frame) {
```

```
    Object value = frame.operandStack.pop();
    frame.operandStack.push(value);
    frame.operandStack.push(value);

    //Log.opcode(frame.getCurrentOpCode());
}

public static void dup_x1(final Frame frame) {
    Object value1 = frame.operandStack.pop();
    Object value2 = frame.operandStack.pop();
    frame.operandStack.push(value1);
    frame.operandStack.push(value2);
    frame.operandStack.push(value1);

    //Log.opcode(frame.getCurrentOpCode());
}

public static void dup_x2(final Frame frame) {
    Object value1 = frame.operandStack.pop();
    Object value2 = frame.operandStack.pop();
    Object value3 = frame.operandStack.pop();
    frame.operandStack.push(value1);
    frame.operandStack.push(value3);
    frame.operandStack.push(value2);
    frame.operandStack.push(value1);

    //Log.opcode(frame.getCurrentOpCode());
}

public static void dup2(final Frame frame) {
    Object value1 = frame.operandStack.pop();
    Object value2 = frame.operandStack.pop();
    frame.operandStack.push(value2);
    frame.operandStack.push(value1);
    frame.operandStack.push(value2);
    frame.operandStack.push(value1);

    //Log.opcode(frame.getCurrentOpCode());
}

public static void dup2_x1(final Frame frame) {
    Object value1 = frame.operandStack.pop();
```

```
        Object value2 = frame.operandStack.pop();
        Object value3 = frame.operandStack.pop();
        frame.operandStack.push(value2);
        frame.operandStack.push(value1);
        frame.operandStack.push(value3);
        frame.operandStack.push(value2);
        frame.operandStack.push(value1);

        //Log.opcode(frame.getCurrentOpCode());
    }

    public static void dup2_x2(final Frame frame) {
        Object value1 = frame.operandStack.pop();
        Object value2 = frame.operandStack.pop();
        Object value3 = frame.operandStack.pop();
        Object value4 = frame.operandStack.pop();
        frame.operandStack.push(value2);
        frame.operandStack.push(value1);
        frame.operandStack.push(value4);
        frame.operandStack.push(value3);
        frame.operandStack.push(value2);
        frame.operandStack.push(value1);

        //Log.opcode(frame.getCurrentOpCode());
    }

}

62:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\stack\Pop.java
 */
package io.nuls.contract.vm.instructions.stack;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Pop {

    public static void pop(final Frame frame) {
        Object value = frame.operandStack.pop();

        //Log.opcode(frame.getCurrentOpCode(), value);
```

```
    }

    public static void pop2(final Frame frame) {
        Object value1 = frame.operandStack.pop();
        Object value2 = frame.operandStack.pop();

        //Log.opcode(frame.getCurrentOpCode(), value1, value2);
    }

}
```

63:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stack\Swap.java

```
 */
package io.nuls.contract.vm.instructions.stack;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Swap {

    public static void swap(final Frame frame) {
        Object value1 = frame.operandStack.pop();
        Object value2 = frame.operandStack.pop();
        frame.operandStack.push(value1);
        frame.operandStack.push(value2);

        //Log.opcode(frame.getCurrentOpCode(), value1, value2);
    }

}
```

64:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stores\Astore.java

```
 */
package io.nuls.contract.vm.instructions.stores;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class Astore {
```

```
    public static void astore(final Frame frame) {
        int index = frame.varInsnNode().var;
        ObjectRef objectRef = frame.operandStack.popRef();
        frame.localVariables.setRef(index, objectRef);

        //Log.result(frame.getCurrentOpCode(), objectRef, index);
    }

}
```

65:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stores\Dstore.java

```
 */
package io.nuls.contract.vm.instructions.stores;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Dstore {

    public static void dstore(final Frame frame) {
        int index = frame.varInsnNode().var;
        double value = frame.operandStack.popDouble();
        frame.localVariables.setDouble(index, value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

66:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stores\Fstore.java

```
 */
package io.nuls.contract.vm.instructions.stores;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Fstore {

    public static void fstore(final Frame frame) {
```

```
        int index = frame.varInsnNode().var;
        float value = frame.operandStack.popFloat();
        frame.localVariables.setFloat(index, value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

67:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stores\Istore.java

```
 */
package io.nuls.contract.vm.instructions.stores;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Istore {

    public static void istore(final Frame frame) {
        int index = frame.varInsnNode().var;
        int value = frame.operandStack.popInt();
        frame.localVariables.setInt(index, value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

68:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stores\Lstore.java

```
 */
package io.nuls.contract.vm.instructions.stores;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Lstore {

    public static void lstore(final Frame frame) {
        int index = frame.varInsnNode().var;
        long value = frame.operandStack.popLong();
```

```
        frame.localVariables.setLong(index, value);

        //Log.result(frame.getCurrentOpCode(), value, index);
    }

}
```

69:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\instructions\stores\Xastore.java

```
 */
package io.nuls.contract.vm.instructions.stores;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.util.Log;

public class Xastore {

    public static void iastore(final Frame frame) {
        int value = frame.operandStack.popInt();
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        frame.heap.putArray(arrayRef, index, value);

        //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
    }

    public static void lastore(final Frame frame) {
        long value = frame.operandStack.popLong();
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        frame.heap.putArray(arrayRef, index, value);

        //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
    }
```

```
public static void fastore(final Frame frame) {
    float value = frame.operandStack.popFloat();
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
    }
    frame.heap.putArray(arrayRef, index, value);

    //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
}

public static void dastore(final Frame frame) {
    double value = frame.operandStack.popDouble();
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
    }
    frame.heap.putArray(arrayRef, index, value);

    //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
}

public static void aastore(final Frame frame) {
    ObjectRef value = frame.operandStack.popRef();
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
    }
    frame.heap.putArray(arrayRef, index, value);

    //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
}

public static void bastore(final Frame frame) {
    int i = frame.operandStack.popInt();
    int index = frame.operandStack.popInt();
    ObjectRef arrayRef = frame.operandStack.popRef();
    if (!frame.checkArray(arrayRef, index)) {
        return;
```

```
        }
        if (arrayRef.getVariableType().getComponentType().isBoolean()) {
            boolean value = i == 1 ? true : false;
            frame.heap.putArray(arrayRef, index, value);
            //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
        } else {
            byte value = (byte) i;
            frame.heap.putArray(arrayRef, index, value);
            //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
        }
    }

    public static void castore(final Frame frame) {
        char value = frame.operandStack.popChar();
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        frame.heap.putArray(arrayRef, index, value);

        //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
    }

    public static void sastore(final Frame frame) {
        short value = frame.operandStack.popShort();
        int index = frame.operandStack.popInt();
        ObjectRef arrayRef = frame.operandStack.popRef();
        if (!frame.checkArray(arrayRef, index)) {
            return;
        }
        frame.heap.putArray(arrayRef, index, value);

        //Log.result(frame.getCurrentOpCode(), arrayRef, index, value);
    }

}

70:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\LocalVariables.java
 */
package io.nuls.contract.vm;
```

```
public class LocalVariables {

    private int maxLocals;

    private Object[] localVariables;

    public LocalVariables(int maxLocals, Object[] args) {
        this.maxLocals = maxLocals;
        this.localVariables = new Object[maxLocals];
        if (args != null) {
            System.arraycopy(args, 0, this.localVariables, 0, args.length);
        }
    }

    public int getInt(int index) {
        Object object = this.localVariables[index];
        if (object instanceof Boolean) {
            return (boolean) object ? 1 : 0;
        } else if (object instanceof Byte) {
            return (byte) object;
        } else if (object instanceof Character) {
            return (char) object;
        } else if (object instanceof Short) {
            return (short) object;
        } else {
            return (int) object;
        }
    }

    public void setInt(int index, int value) {
        this.localVariables[index] = value;
    }

    public long getLong(int index) {
        return (long) this.localVariables[index];
    }

    public void setLong(int index, long value) {
        this.localVariables[index] = value;
    }
```

```
    public float getFloat(int index) {
        return (float) this.localVariables[index];
    }

    public void setFloat(int index, float value) {
        this.localVariables[index] = value;
    }

    public double getDouble(int index) {
        return (double) this.localVariables[index];
    }

    public void setDouble(int index, double value) {
        this.localVariables[index] = value;
    }

    public ObjectRef getRef(int index) {
        return (ObjectRef) this.localVariables[index];
    }

    public void setRef(int index, ObjectRef value) {
        this.localVariables[index] = value;
    }

}

71:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\MethodArea.java
 */
package io.nuls.contract.vm;

import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.ClassCodeLoader;
import io.nuls.contract.vm.code.FieldCode;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.util.Constants;
import io.nuls.contract.vm.util.Log;
import org.apache.commons.lang3.StringUtils;

import java.util.HashMap;
import java.util.Map;
```

```
public class MethodArea {

    public static final Map<String, ClassCode> INIT_CLASS_CODES = new HashMap<>(1024);

    public static final Map<String, MethodCode> INIT_METHOD_CODES = new
HashMap<>(1024);

    private VM vm;

    private final Map<String, ClassCode> classCodes = new HashMap<>(1024);

    private final Map<String, MethodCode> methodCodes = new HashMap<>(1024);

    public MethodArea() {
    }

    public void setVm(VM vm) {
        this.vm = vm;
    }

    public MethodCode loadMethod(String className, String methodName, String methodDesc) {
        String fullName;
        if (StringUtils.isNotEmpty(methodDesc)) {
            fullName = className + "." + methodName + methodDesc;
        } else {
            fullName = className + "." + methodName;
        }
        if (INIT_METHOD_CODES.containsKey(fullName)) {
            return INIT_METHOD_CODES.get(fullName);
        }
        if (methodCodes.containsKey(fullName)) {
            return methodCodes.get(fullName);
        }
        ClassCode classCode = loadClass(className);
        MethodCode methodCode = classCode.getMethodCode(methodName, methodDesc);
        if (methodCode == null && classCode.superName != null) {
            methodCode = loadSuperMethod(classCode.superName, methodName, methodDesc);
        }
        if (methodCode == null) {
            for (String interfaceName : classCode.interfaces) {
                methodCode = loadMethod(interfaceName, methodName, methodDesc);
                if (methodCode != null) {
```

```
                break;
            }
        }
    }
    methodCodes.put(fullName, methodCode);
    return methodCode;
}

private MethodCode loadSuperMethod(String className, String methodName, String
methodDesc) {
    ClassCode classCode = loadClass(className);
    MethodCode methodCode = classCode.getMethodCode(methodName, methodDesc);
    if (methodCode == null && classCode.superName != null) {
        methodCode = loadSuperMethod(classCode.superName, methodName, methodDesc);
    }
    return methodCode;
}

public ClassCode loadClass(String className) {
    if (INIT_CLASS_CODES.containsKey(className)) {
        return INIT_CLASS_CODES.get(className);
    } else {
        if (!this.classCodes.containsKey(className)) {
            ClassCode classCode = ClassCodeLoader.loadFromResource(className);
            loadClassCode(classCode);
        }
        return this.classCodes.get(className);
    }
}

public void loadClassCodes(Map<String, ClassCode> classCodes) {
    if (classCodes != null) {
        for (ClassCode classCode : classCodes.values()) {
            loadClassCode(classCode);
        }
    }
}

public void loadClassCode(ClassCode classCode) {
    String className = classCode.name;
    if (!INIT_CLASS_CODES.containsKey(className)) {
        if (!this.classCodes.containsKey(className)) {
```

```
                this.classCodes.put(className, classCode);
                //Log.loadClass(className);
                clinit(classCode);
            }
        }
    }

    private void clinit(ClassCode classCode) {
        for (FieldCode fieldCode : classCode.fields.values()) {
            if (fieldCode.isStatic && !fieldCode.isFinal) {
                this.vm.heap.putStatic(classCode.name, fieldCode.name,
fieldCode.variableType.getDefaultValue());
            }
        }

        MethodCode methodCode = classCode.getMethodCode(Constants.CLINIT_NAME,
Constants.CLINIT_DESC);
        if (methodCode != null) {
            this.vm.run(methodCode, null, true);
        }
    }

    public Map<String, ClassCode> getClassCodes() {
        return classCodes;
    }

    public Map<String, MethodCode> getMethodCodes() {
        return methodCodes;
    }

}

72:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\MethodArgs.java
 */
package io.nuls.contract.vm;

import io.nuls.contract.vm.code.VariableType;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```
public class MethodArgs {

    public final Object[] frameArgs;

    public final Object[] invokeArgs;

    public final ObjectRef objectRef;

    public MethodArgs(List<VariableType> argsVariableType, OperandStack operandStack,
boolean isStatic) {
        int size = argsVariableType.size();
        List frameList = new ArrayList();
        List invokeList = new ArrayList();
        for (int i = size - 1; i >= 0; i--) {
            VariableType variableType = argsVariableType.get(i);
            if (variableType.isLong() || variableType.isDouble()) {
                frameList.add(operandStack.pop());
            }
            Object value = operandStack.pop();
            frameList.add(value);
            invokeList.add(variableType.getPrimitiveValue(value));
        }
        if (!isStatic) {
            this.objectRef = (ObjectRef) operandStack.pop();
            frameList.add(this.objectRef);
        } else {
            this.objectRef = null;
        }
        Collections.reverse(frameList);
        Collections.reverse(invokeList);
        this.frameArgs = frameList.toArray();
        this.invokeArgs = invokeList.toArray();
    }

}

73:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\io\nuls\contract\sdk\NativeAddress.java
 */
package io.nuls.contract.vm.natives.io.nuls.contract.sdk;
```

```
import io.nuls.contract.sdk.Address;
import io.nuls.contract.vm.*;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.exception.ErrorException;
import io.nuls.contract.vm.exception.RevertException;
import io.nuls.contract.vm.natives.NativeMethod;
import io.nuls.contract.vm.program.ProgramCall;
import io.nuls.contract.vm.program.ProgramResult;
import io.nuls.contract.vm.program.ProgramTransfer;
import io.nuls.contract.vm.program.impl.ProgramInvoke;
import io.nuls.kernel.utils.AddressTool;

import java.math.BigInteger;
import java.util.Arrays;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeAddress {

    public static final String TYPE = "io/nuls/contract/sdk/Address";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case balance:
                if (check) {
                    return SUCCESS;
                } else {
                    return balance(methodCode, methodArgs, frame);
                }
            case transfer:
                if (check) {
                    return SUCCESS;
                } else {
                    return transfer(methodCode, methodArgs, frame);
                }
            case call:
                if (check) {
                    return SUCCESS;
                } else {
                    return call(methodCode, methodArgs, frame);
                }
```

```
            case callWithReturnValue:
                if (check) {
                    return SUCCESS;
                } else {
                    return callWithReturnValue(methodCode, methodArgs, frame);
                }
            case valid:
                if (check) {
                    return SUCCESS;
                } else {
                    return valid(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
    }

    private static BigInteger balance(byte[] address, Frame frame) {
        if (!frame.vm.getRepository().isExist(address)) {
            return BigInteger.ZERO;
        } else {
            return frame.vm.getProgramExecutor().getAccount(address).getBalance();
        }
    }

    public static final String balance = TYPE + "." + "balance" + "()Ljava/math/BigInteger;";

    /**
     * native
     *
     * @see Address#balance()
     */
    private static Result balance(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        String address = frame.heap.runToString(objectRef);
        BigInteger balance = balance(NativeAddress.toBytes(address), frame);
        ObjectRef balanceRef = frame.heap.newBigInteger(balance.toString());
        Result result = NativeMethod.result(methodCode, balanceRef, frame);
        return result;
    }
```

```
    public static final String transfer = TYPE + "." + "transfer" + "(Ljava/math/BigInteger;)V";

    /**
     * native
     *
     * @see Address#transfer(BigInteger)
     */
    private static Result transfer(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef addressRef = methodArgs.objectRef;
        ObjectRef valueRef = (ObjectRef) methodArgs.invokeArgs[0];
        String address = frame.heap.runToString(addressRef);
        BigInteger value = frame.heap.toBigInteger(valueRef);
        byte[] from = frame.vm.getProgramInvoke().getContractAddress();
        byte[] to = NativeAddress.toBytes(address);
        if (Arrays.equals(from, to)) {
            throw new ErrorException(String.format("Cannot transfer from %s to %s",
NativeAddress.toString(from), address), frame.vm.getGasUsed(), null);
        }
        checkBalance(from, value, frame);

        frame.vm.addGasUsed(GasCost.TRANSFER);

        if (frame.heap.existContract(to)) {
            //String address;
            String methodName = "_payable";
            String methodDesc = "()V";
            String[][] args = null;
            //BigInteger value;
            call(address, methodName, methodDesc, args, value, frame);
        } else {
            frame.vm.getProgramExecutor().getAccount(from).addBalance(value.negate());
            ProgramTransfer programTransfer = new ProgramTransfer(from, to, value);
            frame.vm.getTransfers().add(programTransfer);
        }

        Result result = NativeMethod.result(methodCode, null, frame);
        return result;
    }

    public static final String call = TYPE + "." + "call" +
```

```
"(Ljava/lang/String;Ljava/lang/String;[[Ljava/lang/String;Ljava/math/BigInteger;)V";

    /**
     * native
     *
     * @see Address#call(String, String, String[][], BigInteger)
     */
    private static Result call(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        return call(methodCode, methodArgs, frame, false);
    }

    private static Result call(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean returnResult) {
        ObjectRef addressRef = methodArgs.objectRef;
        ObjectRef methodNameRef = (ObjectRef) methodArgs.invokeArgs[0];
        ObjectRef methodDescRef = (ObjectRef) methodArgs.invokeArgs[1];
        ObjectRef argsRef = (ObjectRef) methodArgs.invokeArgs[2];
        ObjectRef valueRef = (ObjectRef) methodArgs.invokeArgs[3];

        String address = frame.heap.runToString(addressRef);
        String methodName = frame.heap.runToString(methodNameRef);
        String methodDesc = frame.heap.runToString(methodDescRef);
        String[][] args = getArgs(argsRef, frame);
        BigInteger value = frame.heap.toBigInteger(valueRef);
        if (value == null) {
            value = BigInteger.ZERO;
        }

        String callResult = call(address, methodName, methodDesc, args, value, frame);
        Object resultValue = null;
        if (returnResult) {
            resultValue = frame.heap.newString(callResult);
        }

        Result result = NativeMethod.result(methodCode, resultValue, frame);
        return result;
    }

    public static final String callWithReturnValue = TYPE + "." + "callWithReturnValue" +
"(Ljava/lang/String;Ljava/lang/String;[[Ljava/lang/String;Ljava/math/BigInteger;)Ljava/lang/String;";

    private static Result callWithReturnValue(MethodCode methodCode, MethodArgs methodArgs,
```

```
Frame frame) {
        return call(methodCode, methodArgs, frame, true);
    }

    private static String[][] getArgs(ObjectRef argsRef, Frame frame) {
        if (argsRef == null) {
            return null;
        }

        int length = argsRef.getDimensions()[0];
        String[][] array = new String[length][0];
        for (int i = 0; i < length; i++) {
            ObjectRef objectRef = (ObjectRef) frame.heap.getArray(argsRef, i);
            String[] ss = (String[]) frame.heap.getObject(objectRef);
            array[i] = ss;
        }

        return array;
    }

    private static String call(String address, String methodName, String methodDesc, String[][] args,
BigInteger value, Frame frame) {
        if (value.compareTo(BigInteger.ZERO) < 0) {
            throw new ErrorException(String.format("amount less than zero, value=%s", value),
frame.vm.getGasUsed(), null);
        }

        ProgramInvoke programInvoke = frame.vm.getProgramInvoke();
        ProgramCall programCall = new ProgramCall();
        programCall.setNumber(programInvoke.getNumber());
        programCall.setSender(programInvoke.getContractAddress());
        programCall.setValue(value != null ? value : BigInteger.ZERO);
        programCall.setGasLimit(programInvoke.getGasLimit() - frame.vm.getGasUsed());
        programCall.setPrice(programInvoke.getPrice());
        programCall.setContractAddress(NativeAddress.toBytes(address));
        programCall.setMethodName(methodName);
        programCall.setMethodDesc(methodDesc);
        programCall.setArgs(args);
        programCall.setEstimateGas(programInvoke.isEstimateGas());
        programCall.setInternalCall(true);

        if (programCall.getValue().compareTo(BigInteger.ZERO) > 0) {
```

```
            checkBalance(programCall.getSender(), programCall.getValue(), frame);
frame.vm.getProgramExecutor().getAccount(programCall.getSender()).addBalance(programCall.g
etValue().negate());
            ProgramTransfer programTransfer = new ProgramTransfer(programCall.getSender(),
programCall.getContractAddress(), programCall.getValue());
            frame.vm.getTransfers().add(programTransfer);
        }

        ProgramResult programResult = frame.vm.getProgramExecutor().call(programCall);

        frame.vm.addGasUsed(programResult.getGasUsed());
        if (programResult.isSuccess()) {
            frame.vm.getTransfers().addAll(programResult.getTransfers());
            frame.vm.getEvents().addAll(programResult.getEvents());
            return programResult.getResult();
        } else if (programResult.isError()) {
            throw new ErrorException(programResult.getErrorMessage(),
programResult.getGasUsed(), programResult.getStackTrace());
        } else if (programResult.isRevert()) {
            throw new RevertException(programResult.getErrorMessage(),
programResult.getStackTrace());
        } else {
            throw new RuntimeException("error contract status");
        }

    }

    private static void checkBalance(byte[] address, BigInteger value, Frame frame) {
        if (value == null || value.compareTo(BigInteger.ZERO) <= 0) {
            throw new ErrorException(String.format("transfer amount error, value=%s", value),
frame.vm.getGasUsed(), null);
        }
        BigInteger balance = frame.vm.getProgramExecutor().getAccount(address).getBalance();
        if (balance.compareTo(value) < 0) {
            if (frame.vm.getProgramContext().isEstimateGas()) {
                balance = value;
            } else {
                throw new ErrorException(String.format("contract[%s] not enough balance",
toString(address)), frame.vm.getGasUsed(), null);
            }
        }
    }
```

```
public static final String valid = TYPE + "." + "valid" + "(Ljava/lang/String;)V";

/**
 * native
 *
 * @see Address#valid(String)
 */
private static Result valid(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    ObjectRef objectRef = (ObjectRef) methodArgs.invokeArgs[0];
    String str = frame.heap.runToString(objectRef);
    boolean valided = validAddress(str);
    if (!valided) {
        frame.throwRuntimeException(String.format("address[%s] error", str));
    }
    Result result = NativeMethod.result(methodCode, null, frame);
    return result;
}

public static String toString(byte[] bytes) {
    if (bytes == null) {
        return null;
    }
    try {
        return AddressTool.getStringAddressByBytes(bytes);
    } catch (Exception e) {
        throw new RuntimeException("address error", e);
    }
}

public static byte[] toBytes(String str) {
    if (str == null) {
        return null;
    }
    try {
        return AddressTool.getAddress(str);
    } catch (Exception e) {
        throw new RuntimeException("address error", e);
    }
}

public static boolean validAddress(String str) {
```

```
        return AddressTool.validAddress(str);
    }

}
```

74:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\natives\io\nuls\contract\sdk\NativeBlock.java

```
 */
package io.nuls.contract.vm.natives.io.nuls.contract.sdk;

import io.nuls.contract.entity.BlockHeaderDto;
import io.nuls.contract.sdk.Block;
import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.natives.NativeMethod;
import org.spongycastle.util.encoders.Hex;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeBlock {

    public static final String TYPE = "io/nuls/contract/sdk/Block";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame frame, boolean check) {
        switch (methodCode.fullName) {
            case getBlockHeader:
                if (check) {
                    return SUCCESS;
                } else {
                    return getBlockHeader(methodCode, methodArgs, frame);
                }
            case currentBlockHeader:
                if (check) {
                    return SUCCESS;
                } else {
                    return currentBlockHeader(methodCode, methodArgs, frame);
                }
```

```
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
    }

    public static final String getBlockHeader = TYPE + "." + "getBlockHeader" +
"(J)Lio/nuls/contract/sdk/BlockHeader;";

    /**
     * native
     *
     * @see Block#getBlockHeader(long)
     */
    private static Result getBlockHeader(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
        long blockNumber = (long) methodArgs.invokeArgs[0];
        ObjectRef objectRef = getBlockHeader(blockNumber, frame);
        Result result = NativeMethod.result(methodCode, objectRef, frame);
        return result;
    }

    public static final String currentBlockHeader = TYPE + "." + "currentBlockHeader" +
"()Lio/nuls/contract/sdk/BlockHeader;";

    /**
     * native
     *
     * @see Block#currentBlockHeader()
     */
    private static Result currentBlockHeader(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
        long blockNumber = frame.vm.getProgramInvoke().getNumber();
        ObjectRef objectRef = getBlockHeader(blockNumber + 1, frame);
        Result result = NativeMethod.result(methodCode, objectRef, frame);
        return result;
    }

    private static ObjectRef getBlockHeader(long blockNumber, Frame frame) {

        String fieldName = "BlockHeader$" + blockNumber;
        Object object = frame.heap.getStatic(VariableType.BLOCK_HEADER_TYPE.getType(),
```

```
fieldName);
        if (object != null) {
            return (ObjectRef) object;
        }

        BlockHeaderDto blockHeaderDto = frame.vm.getBlockHeader(blockNumber);

        if (blockHeaderDto != null) {
            ObjectRef objectRef = frame.heap.newObject(VariableType.BLOCK_HEADER_TYPE);
            frame.heap.putField(objectRef, "hash",
frame.heap.newString(blockHeaderDto.getHash()));
            frame.heap.putField(objectRef, "time", blockHeaderDto.getTime());
            frame.heap.putField(objectRef, "height", blockHeaderDto.getHeight());
            frame.heap.putField(objectRef, "txCount", blockHeaderDto.getTxCount());
            ObjectRef packingAddress = null;
            if (blockHeaderDto.getPackingAddress() != null) {
                packingAddress =
frame.heap.newAddress(NativeAddress.toString(blockHeaderDto.getPackingAddress()));
            }
            frame.heap.putField(objectRef, "packingAddress", packingAddress);
            String stateRoot = null;
            if (blockHeaderDto.getStateRoot() != null) {
                stateRoot = Hex.toHexString(blockHeaderDto.getStateRoot());
            }
            frame.heap.putField(objectRef, "stateRoot", frame.heap.newString(stateRoot));
            frame.heap.putStatic(VariableType.BLOCK_HEADER_TYPE.getType(), fieldName,
objectRef);
            return objectRef;
        }

        return null;
    }

}

75:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\io\nuls\contract\sdk\NativeMsg.java
 */
package io.nuls.contract.vm.natives.io.nuls.contract.sdk;

import io.nuls.contract.sdk.Msg;
import io.nuls.contract.vm.Frame;
```

```
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeMsg {

    public static final String TYPE = "io/nuls/contract/sdk/Msg";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case gasleft:
                if (check) {
                    return SUCCESS;
                } else {
                    return gasleft(methodCode, methodArgs, frame);
                }
            case sender:
                if (check) {
                    return SUCCESS;
                } else {
                    return sender(methodCode, methodArgs, frame);
                }
            case value:
                if (check) {
                    return SUCCESS;
                } else {
                    return value(methodCode, methodArgs, frame);
                }
            case gasprice:
                if (check) {
                    return SUCCESS;
                } else {
                    return gasprice(methodCode, methodArgs, frame);
                }
            case address:
                if (check) {
                    return SUCCESS;
                } else {
```

```
                return address(methodCode, methodArgs, frame);
            }
        default:
            frame.nonsupportMethod(methodCode);
            return null;
      }
   }

   public static final String gasleft = TYPE + "." + "gasleft" + "()J";

   /**
    * native
    *
    * @see Msg#gasleft()
    */
   private static Result gasleft(MethodCode methodCode, MethodArgs methodArgs, Frame frame)
{
      Result result = NativeMethod.result(methodCode, frame.vm.getGasLeft(), frame);
      return result;
   }

   public static final String sender = TYPE + "." + "sender" + "()Lio/nuls/contract/sdk/Address;";

   /**
    * native
    *
    * @see Msg#sender()
    */
   private static Result sender(MethodCode methodCode, MethodArgs methodArgs, Frame frame)
{
      Result result = NativeMethod.result(methodCode,
frame.vm.getProgramContext().getSender(), frame);
      return result;
   }

   public static final String value = TYPE + "." + "value" + "()Ljava/math/BigInteger;";

   /**
    * native
    *
    * @see Msg#value()
    */
```

```
    private static Result value(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        Result result = NativeMethod.result(methodCode, frame.vm.getProgramContext().getValue(),
frame);
        return result;
    }

    public static final String gasprice = TYPE + "." + "gasprice" + "()J";

    /**
     * native
     *
     * @see Msg#gasprice()
     */
    private static Result gasprice(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        Result result = NativeMethod.result(methodCode,
frame.vm.getProgramContext().getGasPrice(), frame);
        return result;
    }

    public static final String address = TYPE + "." + "address" + "()Lio/nuls/contract/sdk/Address;";

    /**
     * native
     *
     * @see Msg#address()
     */
    private static Result address(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        Result result = NativeMethod.result(methodCode,
frame.vm.getProgramContext().getAddress(), frame);
        return result;
    }

}

76:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\io\nuls\contract\sdk\NativeUtils.java
 */
package io.nuls.contract.vm.natives.io.nuls.contract.sdk;

import io.nuls.contract.sdk.Event;
```

```
import io.nuls.contract.sdk.Utils;
import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.FieldCode;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.exception.ErrorException;
import io.nuls.contract.vm.natives.NativeMethod;
import io.nuls.contract.vm.util.JsonUtils;

import java.util.LinkedHashMap;
import java.util.Map;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;
import static io.nuls.contract.vm.util.Utils.hashMapInitialCapacity;

public class NativeUtils {

    public static final String TYPE = "io/nuls/contract/sdk/Utils";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case revert:
                if (check) {
                    return SUCCESS;
                } else {
                    return revert(methodCode, methodArgs, frame);
                }
            case emit:
                if (check) {
                    return SUCCESS;
                } else {
                    return emit(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
```

```
    }

    public static final String revert = TYPE + "." + "revert" + "(Ljava/lang/String;)V";

    /**
     * native
     *
     * @see Utils#revert(String)
     */
    private static Result revert(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        ObjectRef objectRef = (ObjectRef) methodArgs.invokeArgs[0];
        String errorMessage = null;
        if (objectRef != null) {
            errorMessage = frame.heap.runToString(objectRef);
        }
        throw new ErrorException(errorMessage, frame.vm.getGasUsed(), null);
    }

    public static final String emit = TYPE + "." + "emit" + "(Lio/nuls/contract/sdk/Event;)V";

    /**
     * native
     *
     * @see Utils#emit(Event)
     */
    private static Result emit(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        ObjectRef objectRef = (ObjectRef) methodArgs.invokeArgs[0];
        //String str = frame.heap.runToString(objectRef);
        ClassCode classCode =
frame.methodArea.loadClass(objectRef.getVariableType().getType());
        Map<String, Object> jsonMap = toJson(objectRef, frame);
        EventJson eventJson = new EventJson();
        eventJson.setContractAddress(frame.vm.getProgramInvoke().getAddress());
        eventJson.setBlockNumber(frame.vm.getProgramInvoke().getNumber() + 1);
        eventJson.setEvent(classCode.simpleName);
        eventJson.setPayload(jsonMap);
        String json = JsonUtils.toJson(eventJson);
        frame.vm.getEvents().add(json);
        Result result = NativeMethod.result(methodCode, null, frame);
        return result;
    }
```

```
    private static Map<String, Object> toJson(ObjectRef objectRef, Frame frame) {
        if (objectRef == null) {
            return null;
        }

        ClassCode classCode =
frame.methodArea.loadClass(objectRef.getVariableType().getType());
        Map<String, Object> map = frame.heap.getFields(objectRef);
        Map<String, Object> jsonMap = new
LinkedHashMap<>(hashMapInitialCapacity(map.size()));
        for (Map.Entry<String, Object> entry : map.entrySet()) {
            String name = entry.getKey();
            FieldCode fieldCode = classCode.fields.get(name);
            if (fieldCode != null && !fieldCode.isSynthetic) {
                Object value = entry.getValue();
                jsonMap.put(name, toJson(fieldCode, value, frame));
            }
        }
        return jsonMap;
    }

    private static Object toJson(FieldCode fieldCode, Object value, Frame frame) {
        VariableType variableType = fieldCode.variableType;
        if (value == null) {
            return null;
        } else if (variableType.isPrimitive()) {
            return variableType.getPrimitiveValue(value);
        } else if (variableType.isArray()) {
            ObjectRef ref = (ObjectRef) value;
            if (variableType.isPrimitiveType() && variableType.getDimensions() == 1) {
                return frame.heap.getObject(ref);
            } else {
                int length = ref.getDimensions()[0];
                Object[] array = new Object[length];
                for (int i = 0; i < length; i++) {
                    Object item = frame.heap.getArray(ref, i);
                    if (item != null) {
                        ObjectRef itemRef = (ObjectRef) item;
                        item = frame.heap.runToString(itemRef);
                    }
                    array[i] = item;
                }
```

```
            return array;
        }
    } else {
        ObjectRef ref = (ObjectRef) value;
        return frame.heap.runToString(ref);
    }
}

static class EventJson {

    private String contractAddress;

    private long blockNumber;

    private String event;

    private Map<String, Object> payload;

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public long getBlockNumber() {
        return blockNumber;
    }

    public void setBlockNumber(long blockNumber) {
        this.blockNumber = blockNumber;
    }

    public String getEvent() {
        return event;
    }

    public void setEvent(String event) {
        this.event = event;
    }
```

```
        public Map<String, Object> getPayload() {
            return payload;
        }

        public void setPayload(Map<String, Object> payload) {
            this.payload = payload;
        }

    }

}

77:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeAbstractStringBuilder.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeAbstractStringBuilder {

    public static final String TYPE = "java/lang/AbstractStringBuilder";

    public static final String appendD = TYPE + "." + "append" +
"(D)Ljava/lang/AbstractStringBuilder;";

    public static final String appendF = TYPE + "." + "append" +
"(F)Ljava/lang/AbstractStringBuilder;";

    public static Result override(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean check) {
        switch (methodCode.fullName) {
            case appendD:
                if (check) {
                    return SUCCESS;
```

```
            } else {
                return append(methodCode, methodArgs, frame);
            }
        case appendF:
            if (check) {
                return SUCCESS;
            } else {
                return append(methodCode, methodArgs, frame);
            }
        default:
            return null;
      }
   }

   /**
    * override
    *
    * @see AbstractStringBuilder#append(float)
    * @see AbstractStringBuilder#append(double)
    */
   private static Result append(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
       ObjectRef objectRef = methodArgs.objectRef;
       Object a = methodArgs.invokeArgs[0];
       ObjectRef ref = frame.heap.newString(a.toString());
       MethodCode append = frame.methodArea.loadMethod(TYPE, "append",
"(Ljava/lang/String;)Ljava/lang/AbstractStringBuilder;");
       frame.vm.run(append, new Object[]{objectRef, ref}, false);
       Result result = NativeMethod.result(methodCode, objectRef, frame);
       return result;
   }

}

78:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeCharacter.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.Result;
```

```
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeCharacter {

    public static final String TYPE = "java/lang/Character";

    public static final String digit = TYPE + "." + "digit" + "(II)I";

    public static Result override(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean check) {
        switch (methodCode.fullName) {
            case digit:
                if (check) {
                    return SUCCESS;
                } else {
                    return digit(methodCode, methodArgs, frame);
                }
            default:
                return null;
        }
    }

    /**
     * override
     *
     * @see Character#digit(int, int)
     */
    private static Result digit(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        int codePoint = (int) methodArgs.invokeArgs[0];
        int radix = (int) methodArgs.invokeArgs[1];
        int i = Character.digit(codePoint, radix);
        Result result = NativeMethod.result(methodCode, i, frame);
        return result;
    }

}
```

79:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeClass.java

```
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.natives.NativeMethod;

import java.util.List;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeClass {

    public static final String TYPE = "java/lang/Class";

    public static Result override(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean check) {
        switch (methodCode.fullName) {
            case getInterfaces:
                if (check) {
                    return SUCCESS;
                } else {
                    return getInterfaces(methodCode, methodArgs, frame);
                }
            default:
                return null;
        }
    }

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case getPrimitiveClass:
                if (check) {
                    return SUCCESS;
                } else {
                    return getPrimitiveClass(methodCode, methodArgs, frame);
```

```
    }
case getComponentType:
    if (check) {
        return SUCCESS;
    } else {
        return getComponentType(methodCode, methodArgs, frame);
    }
case isArray:
    if (check) {
        return SUCCESS;
    } else {
        return isArray(methodCode, methodArgs, frame);
    }
case isPrimitive:
    if (check) {
        return SUCCESS;
    } else {
        return isPrimitive(methodCode, methodArgs, frame);
    }
case isInterface:
    if (check) {
        return SUCCESS;
    } else {
        return isInterface(methodCode, methodArgs, frame);
    }
case desiredAssertionStatus0:
    if (check) {
        return SUCCESS;
    } else {
        return desiredAssertionStatus0(methodCode, methodArgs, frame);
    }
case getGenericSignature0:
    if (check) {
        return SUCCESS;
    } else {
        return getGenericSignature0(methodCode, methodArgs, frame);
    }
case getName0:
    if (check) {
        return SUCCESS;
    } else {
        return getName0(methodCode, methodArgs, frame);
```

```
            }
        default:
            frame.nonsupportMethod(methodCode);
            return null;
    }
  }

  public static final String getPrimitiveClass = TYPE + "." + "getPrimitiveClass" +
"(Ljava/lang/String;)Ljava/lang/Class;";

  /**
   * native
   *
   * @see Class#getPrimitiveClass(String)
   */
  private static Result getPrimitiveClass(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
      ObjectRef objectRef = (ObjectRef) methodArgs.invokeArgs[0];
      String name = frame.heap.runToString(objectRef);
      VariableType variableType = VariableType.valueOf(name);
      ObjectRef classRef = frame.heap.getClassRef(variableType.getDesc());
      Result result = NativeMethod.result(methodCode, classRef, frame);
      return result;
  }

  public static final String getComponentType = TYPE + "." + "getComponentType" +
"()Ljava/lang/Class;";

  /**
   * native
   *
   * @see Class#getComponentType()
   */
  private static Result getComponentType(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
      ObjectRef objectRef = methodArgs.objectRef;
      VariableType variableType;
      if (objectRef.getVariableType().isArray()) {
          variableType = objectRef.getVariableType();
      } else {
          variableType = VariableType.valueOf(objectRef.getRef());
      }
```

```
        ObjectRef classRef = null;
        if (variableType.isArray()) {
            classRef = frame.heap.getClassRef(variableType.getComponentType().getDesc());
        }
        Result result = NativeMethod.result(methodCode, classRef, frame);
        return result;
    }

    public static final String isArray = TYPE + "." + "isArray" + "()Z";

    /**
     * native
     *
     * @see Class#isArray()
     */
    private static Result isArray(MethodCode methodCode, MethodArgs methodArgs, Frame frame)
{
        ObjectRef objectRef = methodArgs.objectRef;
        VariableType variableType;
        if (objectRef.getVariableType().isArray()) {
            variableType = objectRef.getVariableType();
        } else {
            variableType = VariableType.valueOf(objectRef.getRef());
        }
        boolean b = variableType.isArray();
        Result result = NativeMethod.result(methodCode, b, frame);
        return result;
    }

    public static final String isPrimitive = TYPE + "." + "isPrimitive" + "()Z";

    /**
     * native
     *
     * @see Class#isPrimitive()
     */
    private static Result isPrimitive(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        VariableType variableType;
        if (objectRef.getVariableType().isArray()) {
            variableType = objectRef.getVariableType();
```

```
        } else {
            variableType = VariableType.valueOf(objectRef.getRef());
        }
        boolean b = variableType.isPrimitive();
        Result result = NativeMethod.result(methodCode, b, frame);
        return result;
    }

    public static final String isInterface = TYPE + "." + "isInterface" + "()Z";

    /**
     * native
     *
     * @see Class#isInterface()
     */
    private static Result isInterface(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        VariableType variableType;
        if (objectRef.getVariableType().isArray()) {
            variableType = objectRef.getVariableType();
        } else {
            variableType = VariableType.valueOf(objectRef.getRef());
        }
        boolean b = false;
        if (!variableType.isArray() && !variableType.isPrimitiveType()) {
            ClassCode classCode = frame.methodArea.loadClass(variableType.getType());
            b = classCode.isInterface;
        }
        Result result = NativeMethod.result(methodCode, b, frame);
        return result;
    }

    public static final String desiredAssertionStatus0 = TYPE + "." + "desiredAssertionStatus0" +
"(Ljava/lang/Class;)Z";

    /**
     * native
     *
     * @see Class#desiredAssertionStatus0(Class)
     */
    private static Result desiredAssertionStatus0(MethodCode methodCode, MethodArgs
```

```
methodArgs, Frame frame) {
        boolean status = false;
        Result result = NativeMethod.result(methodCode, status, frame);
        return result;
    }

    public static final String getGenericSignature0 = TYPE + "." + "getGenericSignature0" +
"()Ljava/lang/String;";

    /**
     * native
     *
     * @see Class#getGenericSignature0()
     */
    private static Result getGenericSignature0(MethodCode methodCode, MethodArgs
methodArgs, Frame frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        VariableType variableType;
        if (objectRef.getVariableType().isArray()) {
            variableType = objectRef.getVariableType();
        } else {
            variableType = VariableType.valueOf(objectRef.getRef());
        }
        ObjectRef ref = null;
        if (!variableType.isPrimitiveType()) {
            ClassCode classCode = frame.methodArea.loadClass(variableType.getType());
            String signature = classCode.signature;
            ref = frame.heap.newString(signature);
        }
        Result result = NativeMethod.result(methodCode, ref, frame);
        return result;
    }

    public static final String getName0 = TYPE + "." + "getName0" + "()Ljava/lang/String;";

    /**
     * native
     *
     * @see Class#getName0()
     */
    private static Result getName0(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
```

```
        ObjectRef objectRef = methodArgs.objectRef;
        VariableType variableType;
        if (objectRef.getVariableType().isArray()) {
            variableType = objectRef.getVariableType();
        } else {
            variableType = VariableType.valueOf(objectRef.getRef());
        }
        String name;
        if (variableType.isArray()) {
            name = variableType.getDesc();
        } else {
            name = variableType.getType();
            if (name.startsWith("L") && name.endsWith(";")) {
                name = name.substring(1, name.length() - 1);
            }
        }
        name = name.replace('/', '.');
        ObjectRef ref = frame.heap.newString(name);
        Result result = NativeMethod.result(methodCode, ref, frame);
        return result;
    }

    public static final String getInterfaces = TYPE + "." + "getInterfaces" + "()[Ljava/lang/Class;";

    /**
     * override
     *
     * @see Class#getInterfaces()
     */
    private static Result getInterfaces(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        VariableType variableType;
        if (objectRef.getVariableType().isArray()) {
            variableType = objectRef.getVariableType();
        } else {
            variableType = VariableType.valueOf(objectRef.getRef());
        }
        ObjectRef array;
        if (!variableType.isPrimitiveType()) {
            ClassCode classCode = frame.methodArea.loadClass(variableType.getType());
            List<String> interfaces = classCode.interfaces;
```

```
            int length = interfaces.size();
            array = frame.heap.newArray(VariableType.valueOf("[Ljava/lang/Class;"), length);
            for (int i = 0; i < length; i++) {
                String interfaceName = interfaces.get(i);
                VariableType interfaceType = VariableType.valueOf(interfaceName);
                ObjectRef ref = frame.heap.getClassRef(interfaceType.getDesc());
                frame.heap.putArray(array, i, ref);
            }
        } else {
            array = frame.heap.newArray(VariableType.valueOf("[Ljava/lang/Class;"), 0);
        }
        Result result = NativeMethod.result(methodCode, array, frame);
        return result;
    }

}

80:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeDouble.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeDouble {

    public static final String TYPE = "java/lang/Double";

    public static Result override(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean check) {
        switch (methodCode.fullName) {
            case parseDouble:
                if (check) {
                    return SUCCESS;
                } else {
```

```
                return parseDouble(methodCode, methodArgs, frame);
            }
        case toString:
            if (check) {
                return SUCCESS;
            } else {
                return toString(methodCode, methodArgs, frame);
            }
        case toHexString:
            if (check) {
                return SUCCESS;
            } else {
                return toHexString(methodCode, methodArgs, frame);
            }
        default:
            return null;
    }
}

public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
    switch (methodCode.fullName) {
        case doubleToRawLongBits:
            if (check) {
                return SUCCESS;
            } else {
                return doubleToRawLongBits(methodCode, methodArgs, frame);
            }
        case longBitsToDouble:
            if (check) {
                return SUCCESS;
            } else {
                return longBitsToDouble(methodCode, methodArgs, frame);
            }
        default:
            frame.nonsupportMethod(methodCode);
            return null;
    }
}

public static final String doubleToRawLongBits = TYPE + "." + "doubleToRawLongBits" + "(D)J";
```

```
    /**
     * native
     *
     * @see Double#doubleToRawLongBits(double)
     */
    private static Result doubleToRawLongBits(MethodCode methodCode, MethodArgs
methodArgs, Frame frame) {
        double value = (double) methodArgs.invokeArgs[0];
        long bits = Double.doubleToRawLongBits(value);
        Result result = NativeMethod.result(methodCode, bits, frame);
        return result;
    }

    public static final String longBitsToDouble = TYPE + "." + "longBitsToDouble" + "(J)D";

    /**
     * native
     *
     * @see Double#longBitsToDouble(long)
     */
    private static Result longBitsToDouble(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
        long bits = (long) methodArgs.invokeArgs[0];
        double d = Double.longBitsToDouble(bits);
        Result result = NativeMethod.result(methodCode, d, frame);
        return result;
    }

    public static final String parseDouble = TYPE + "." + "parseDouble" + "(Ljava/lang/String;)D";

    /**
     * override
     *
     * @see Double#parseDouble(String)
     */
    private static Result parseDouble(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = (ObjectRef) methodArgs.invokeArgs[0];
        String s = frame.heap.runToString(objectRef);
        double d;
        try {
            d = Double.parseDouble(s);
```

```
        } catch (Exception e) {
            frame.throwNumberFormatException(e.getMessage());
            return null;
        }
        Result result = NativeMethod.result(methodCode, d, frame);
        return result;
    }

    public static final String toString = TYPE + "." + "toString" + "(D)Ljava/lang/String;";

    /**
     * override
     *
     * @see Double#toString(double)
     */
    private static Result toString(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        double d = (double) methodArgs.invokeArgs[0];
        String s = Double.toString(d);
        ObjectRef ref = frame.heap.newString(s);
        Result result = NativeMethod.result(methodCode, ref, frame);
        return result;
    }

    public static final String toHexString = TYPE + "." + "toHexString" + "(D)Ljava/lang/String;";

    /**
     * override
     *
     * @see Double#toHexString(double)
     */
    private static Result toHexString(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        double d = (double) methodArgs.invokeArgs[0];
        String s = Double.toHexString(d);
        ObjectRef ref = frame.heap.newString(s);
        Result result = NativeMethod.result(methodCode, ref, frame);
        return result;
    }

}
```

```
81:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeFloat.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeFloat {

    public static final String TYPE = "java/lang/Float";

    public static Result override(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean check) {
        switch (methodCode.fullName) {
            case parseFloat:
                if (check) {
                    return SUCCESS;
                } else {
                    return parseFloat(methodCode, methodArgs, frame);
                }
            case toString:
                if (check) {
                    return SUCCESS;
                } else {
                    return toString(methodCode, methodArgs, frame);
                }
            case toHexString:
                if (check) {
                    return SUCCESS;
                } else {
                    return toHexString(methodCode, methodArgs, frame);
                }
            default:
                return null;
        }
```

```
    }

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case intBitsToFloat:
                if (check) {
                    return SUCCESS;
                } else {
                    return intBitsToFloat(methodCode, methodArgs, frame);
                }
            case floatToRawIntBits:
                if (check) {
                    return SUCCESS;
                } else {
                    return floatToRawIntBits(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
    }

    public static final String intBitsToFloat = TYPE + "." + "intBitsToFloat" + "(I)F";

    /**
     * native
     *
     * @see Float#intBitsToFloat(int)
     */
    private static Result intBitsToFloat(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        int bits = (int) methodArgs.invokeArgs[0];
        float f = Float.intBitsToFloat(bits);
        Result result = NativeMethod.result(methodCode, f, frame);
        return result;
    }

    public static final String floatToRawIntBits = TYPE + "." + "floatToRawIntBits" + "(F)I";

    /**
     * native
```

```
     *
     * @see Float#floatToRawIntBits(float)
     */
    private static Result floatToRawIntBits(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
        float value = (float) methodArgs.invokeArgs[0];
        int bits = Float.floatToRawIntBits(value);
        Result result = NativeMethod.result(methodCode, bits, frame);
        return result;
    }

    public static final String parseFloat = TYPE + "." + "parseFloat" + "(Ljava/lang/String;)F";

    /**
     * override
     *
     * @see Float#parseFloat(String)
     */
    private static Result parseFloat(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = (ObjectRef) methodArgs.invokeArgs[0];
        String s = frame.heap.runToString(objectRef);
        float f;
        try {
            f = Float.parseFloat(s);
        } catch (Exception e) {
            frame.throwNumberFormatException(e.getMessage());
            return null;
        }
        Result result = NativeMethod.result(methodCode, f, frame);
        return result;
    }

    public static final String toString = TYPE + "." + "toString" + "(F)Ljava/lang/String;";

    /**
     * override
     *
     * @see Float#toString(float)
     */
    private static Result toString(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
```

```
        float f = (float) methodArgs.invokeArgs[0];
        String s = Float.toString(f);
        ObjectRef ref = frame.heap.newString(s);
        Result result = NativeMethod.result(methodCode, ref, frame);
        return result;
    }

    public static final String toHexString = TYPE + "." + "toHexString" + "(F)Ljava/lang/String;";

    /**
     * override
     *
     * @see Float#toHexString(float)
     */
    private static Result toHexString(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        float f = (float) methodArgs.invokeArgs[0];
        String s = Float.toHexString(f);
        ObjectRef ref = frame.heap.newString(s);
        Result result = NativeMethod.result(methodCode, ref, frame);
        return result;
    }

}

82:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeObject.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;
import io.nuls.contract.vm.util.CloneUtils;

import java.util.Map;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;
```

```
public class NativeObject {

    public static final String TYPE = "java/lang/Object";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case getClass:
                if (check) {
                    return SUCCESS;
                } else {
                    return getClass(methodCode, methodArgs, frame);
                }
            case hashCode:
                if (check) {
                    return SUCCESS;
                } else {
                    return hashCode(methodCode, methodArgs, frame);
                }
            case clone:
                if (check) {
                    return SUCCESS;
                } else {
                    return clone(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
    }

    public static final String getClass = TYPE + "." + "getClass" + "()Ljava/lang/Class;";

    /**
     * native
     *
     * @see Object#getClass()
     */
    private static Result getClass(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        ObjectRef classRef = frame.heap.getClassRef(objectRef.getVariableType().getDesc());
```

```
        Result result = NativeMethod.result(methodCode, classRef, frame);
        return result;
    }

    public static final String hashCode = TYPE + "." + "hashCode" + "()I";

    /**
     * native
     *
     * @see Object#hashCode()
     */
    private static Result hashCode(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        int hashCode = objectRef.hashCode();
        Result result = NativeMethod.result(methodCode, hashCode, frame);
        return result;
    }

    public static final String clone = TYPE + "." + "clone" + "()Ljava/lang/Object;";

    /**
     * native
     *
     * @see Object#clone()
     */
    private static Result clone(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        Map<String, Object> fields = frame.heap.getFields(objectRef);
        Map<String, Object> newFields = CloneUtils.clone(fields);
        ObjectRef newRef = frame.heap.newObjectRef(null, objectRef.getDesc(),
objectRef.getDimensions());
        frame.heap.putFields(newRef, newFields);
        Result result = NativeMethod.result(methodCode, newRef, frame);
        return result;
    }

}

83:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeStrictMath.java
 */
```

```
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeStrictMath {

    public static final String TYPE = "java/lang/StrictMath";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case sin:
                if (check) {
                    return SUCCESS;
                } else {
                    return sin(methodCode, methodArgs, frame);
                }
            case cos:
                if (check) {
                    return SUCCESS;
                } else {
                    return cos(methodCode, methodArgs, frame);
                }
            case tan:
                if (check) {
                    return SUCCESS;
                } else {
                    return tan(methodCode, methodArgs, frame);
                }
            case asin:
                if (check) {
                    return SUCCESS;
                } else {
                    return asin(methodCode, methodArgs, frame);
                }
            case acos:
```

```
    if (check) {
        return SUCCESS;
    } else {
        return acos(methodCode, methodArgs, frame);
    }
case atan:
    if (check) {
        return SUCCESS;
    } else {
        return atan(methodCode, methodArgs, frame);
    }
case exp:
    if (check) {
        return SUCCESS;
    } else {
        return exp(methodCode, methodArgs, frame);
    }
case log:
    if (check) {
        return SUCCESS;
    } else {
        return log(methodCode, methodArgs, frame);
    }
case log10:
    if (check) {
        return SUCCESS;
    } else {
        return log10(methodCode, methodArgs, frame);
    }
case sqrt:
    if (check) {
        return SUCCESS;
    } else {
        return sqrt(methodCode, methodArgs, frame);
    }
case cbrt:
    if (check) {
        return SUCCESS;
    } else {
        return cbrt(methodCode, methodArgs, frame);
    }
case IEEEremainder:
```

```
    if (check) {
        return SUCCESS;
    } else {
        return IEEEremainder(methodCode, methodArgs, frame);
    }
case atan2:
    if (check) {
        return SUCCESS;
    } else {
        return atan2(methodCode, methodArgs, frame);
    }
case pow:
    if (check) {
        return SUCCESS;
    } else {
        return pow(methodCode, methodArgs, frame);
    }
case sinh:
    if (check) {
        return SUCCESS;
    } else {
        return sinh(methodCode, methodArgs, frame);
    }
case cosh:
    if (check) {
        return SUCCESS;
    } else {
        return cosh(methodCode, methodArgs, frame);
    }
case tanh:
    if (check) {
        return SUCCESS;
    } else {
        return tanh(methodCode, methodArgs, frame);
    }
case hypot:
    if (check) {
        return SUCCESS;
    } else {
        return hypot(methodCode, methodArgs, frame);
    }
case expm1:
```

```
            if (check) {
                return SUCCESS;
            } else {
                return expm1(methodCode, methodArgs, frame);
            }
        case log1p:
            if (check) {
                return SUCCESS;
            } else {
                return log1p(methodCode, methodArgs, frame);
            }
        default:
            frame.nonsupportMethod(methodCode);
            return null;
    }
}

public static final String sin = TYPE + "." + "sin" + "(D)D";

/**
 * native
 *
 * @see StrictMath#sin(double)
 */
private static Result sin(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.sin(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String cos = TYPE + "." + "cos" + "(D)D";

/**
 * native
 *
 * @see StrictMath#cos(double)
 */
private static Result cos(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.cos(a);
    Result result = NativeMethod.result(methodCode, r, frame);
```

```
    return result;
}

public static final String tan = TYPE + "." + "tan" + "(D)D";

/**
 * native
 *
 * @see StrictMath#tan(double)
 */
private static Result tan(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.tan(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String asin = TYPE + "." + "asin" + "(D)D";

/**
 * native
 *
 * @see StrictMath#asin(double)
 */
private static Result asin(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.asin(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String acos = TYPE + "." + "acos" + "(D)D";

/**
 * native
 *
 * @see StrictMath#acos(double)
 */
private static Result acos(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.acos(a);
    Result result = NativeMethod.result(methodCode, r, frame);
```

```
    return result;
}

public static final String atan = TYPE + "." + "atan" + "(D)D";

/**
 * native
 *
 * @see StrictMath#atan(double)
 */
private static Result atan(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.atan(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String exp = TYPE + "." + "exp" + "(D)D";

/**
 * native
 *
 * @see StrictMath#exp(double)
 */
private static Result exp(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.exp(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String log = TYPE + "." + "log" + "(D)D";

/**
 * native
 *
 * @see StrictMath#log(double)
 */
private static Result log(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.log(a);
    Result result = NativeMethod.result(methodCode, r, frame);
```

```
    return result;
}

public static final String log10 = TYPE + "." + "log10" + "(D)D";

/**
 * native
 *
 * @see StrictMath#log10(double)
 */
private static Result log10(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.log10(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String sqrt = TYPE + "." + "sqrt" + "(D)D";

/**
 * native
 *
 * @see StrictMath#sqrt(double)
 */
private static Result sqrt(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.sqrt(a);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String cbrt = TYPE + "." + "cbrt" + "(D)D";

/**
 * native
 *
 * @see StrictMath#cbrt(double)
 */
private static Result cbrt(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double a = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.cbrt(a);
    Result result = NativeMethod.result(methodCode, r, frame);
```

```
        return result;
    }

    public static final String IEEEremainder = TYPE + "." + "IEEEremainder" + "(DD)D";

    /**
     * native
     *
     * @see StrictMath#IEEEremainder(double, double)
     */
    private static Result IEEEremainder(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
        double f1 = (double) methodArgs.invokeArgs[0];
        double f2 = (double) methodArgs.invokeArgs[1];
        double r = StrictMath.IEEEremainder(f1, f2);
        Result result = NativeMethod.result(methodCode, r, frame);
        return result;
    }

    public static final String atan2 = TYPE + "." + "atan2" + "(DD)D";

    /**
     * native
     *
     * @see StrictMath#atan2(double, double)
     */
    private static Result atan2(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        double y = (double) methodArgs.invokeArgs[0];
        double x = (double) methodArgs.invokeArgs[1];
        double r = StrictMath.atan2(y, x);
        Result result = NativeMethod.result(methodCode, r, frame);
        return result;
    }

    public static final String pow = TYPE + "." + "pow" + "(DD)D";

    /**
     * native
     *
     * @see StrictMath#pow(double, double)
     */
    private static Result pow(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
```

```
    double a = (double) methodArgs.invokeArgs[0];
    double b = (double) methodArgs.invokeArgs[1];
    double r = StrictMath.pow(a, b);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String sinh = TYPE + "." + "sinh" + "(D)D";

/**
 * native
 *
 * @see StrictMath#sinh(double)
 */
private static Result sinh(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double x = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.sinh(x);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String cosh = TYPE + "." + "cosh" + "(D)D";

/**
 * native
 *
 * @see StrictMath#cosh(double)
 */
private static Result cosh(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    double x = (double) methodArgs.invokeArgs[0];
    double r = StrictMath.cosh(x);
    Result result = NativeMethod.result(methodCode, r, frame);
    return result;
}

public static final String tanh = TYPE + "." + "tanh" + "(D)D";

/**
 * native
 *
 * @see StrictMath#tanh(double)
 */
```

```
    private static Result tanh(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        double x = (double) methodArgs.invokeArgs[0];
        double r = StrictMath.tanh(x);
        Result result = NativeMethod.result(methodCode, r, frame);
        return result;
    }

    public static final String hypot = TYPE + "." + "hypot" + "(DD)D";

    /**
     * native
     *
     * @see StrictMath#hypot(double, double)
     */
    private static Result hypot(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        double x = (double) methodArgs.invokeArgs[0];
        double y = (double) methodArgs.invokeArgs[1];
        double r = StrictMath.hypot(x, y);
        Result result = NativeMethod.result(methodCode, r, frame);
        return result;
    }

    public static final String expm1 = TYPE + "." + "expm1" + "(D)D";

    /**
     * native
     *
     * @see StrictMath#expm1(double)
     */
    private static Result expm1(MethodCode methodCode, MethodArgs methodArgs, Frame frame)
{
        double x = (double) methodArgs.invokeArgs[0];
        double r = StrictMath.expm1(x);
        Result result = NativeMethod.result(methodCode, r, frame);
        return result;
    }

    public static final String log1p = TYPE + "." + "log1p" + "(D)D";

    /**
     * native
     *
```

```
     * @see StrictMath#log1p(double)
     */
    private static Result log1p(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        double x = (double) methodArgs.invokeArgs[0];
        double r = StrictMath.log1p(x);
        Result result = NativeMethod.result(methodCode, r, frame);
        return result;
    }

}
```

84:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeString.java

```
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeString {

    public static final String TYPE = "java/lang/String";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame frame, boolean check) {
        switch (methodCode.fullName) {
            case intern:
                if (check) {
                    return SUCCESS;
                } else {
                    return intern(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
```

```
    }

    public static final String intern = TYPE + "." + "intern" + "()Ljava/lang/String;";

    /**
     * native
     *
     * @see String#intern()
     */
    private static Result intern(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        Result result = NativeMethod.result(methodCode, objectRef, frame);
        return result;
    }

}

85:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeSystem.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeSystem {

    public static final String TYPE = "java/lang/System";

    public static final String getProperty = TYPE + "." + "getProperty" +
"(Ljava/lang/String;)Ljava/lang/String;";

    public static final String getProperty_ = TYPE + "." + "getProperty" +
"(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
```

```
frame, boolean check) {
        switch (methodCode.fullName) {
            case arraycopy:
                if (check) {
                    return SUCCESS;
                } else {
                    return arraycopy(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
    }

    public static final String arraycopy = TYPE + "." + "arraycopy" +
"(Ljava/lang/Object;ILjava/lang/Object;II)V";

    /**
     * native
     *
     * @see System#arraycopy(Object, int, Object, int, int)
     */
    private static Result arraycopy(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        Object[] args = methodArgs.invokeArgs;
        ObjectRef srcObjectRef = (ObjectRef) args[0];
        int srcPos = (int) args[1];
        ObjectRef destObjectRef = (ObjectRef) args[2];
        int destPos = (int) args[3];
        int length = (int) args[4];

        if (length > 0 && frame.checkArray(srcObjectRef, srcPos)
                && frame.checkArray(srcObjectRef, srcPos + length - 1)
                && frame.checkArray(destObjectRef, destPos)
                && frame.checkArray(destObjectRef, destPos + length - 1)) {
            frame.heap.arraycopy(srcObjectRef, srcPos, destObjectRef, destPos, length);
        }

        Result result = NativeMethod.result(methodCode, null, frame);
        return result;
    }
```

```
}

86:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\NativeThrowable.java
 */
package io.nuls.contract.vm.natives.java.lang;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.instructions.references.Instanceof;
import io.nuls.contract.vm.natives.NativeMethod;

import java.util.ArrayList;
import java.util.List;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeThrowable {

    public static final String TYPE = "java/lang/Throwable";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case fillInStackTrace:
                if (check) {
                    return SUCCESS;
                } else {
                    return fillInStackTrace(methodCode, methodArgs, frame);
                }
            case getStackTraceDepth:
                if (check) {
                    return SUCCESS;
                } else {
                    return getStackTraceDepth(methodCode, methodArgs, frame);
                }
            case getStackTraceElement:
                if (check) {
```

```
                return SUCCESS;
            } else {
                return getStackTraceElement(methodCode, methodArgs, frame);
            }
        default:
            frame.nonsupportMethod(methodCode);
            return null;
    }
  }

  public static final String fillInStackTrace = TYPE + "." + "fillInStackTrace" +
"(I)Ljava/lang/Throwable;";

  /**
   * native
   *
   * @see Throwable#fillInStackTrace(int)
   */
  private static Result fillInStackTrace(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
      int dummy = (int) methodArgs.invokeArgs[0];
      ObjectRef objectRef = methodArgs.objectRef;

      int size = frame.vm.vmStack.size();
      boolean isThrowable = true;
      List<Frame> frames = new ArrayList<>();
      for (int i = size - 1; i >= 0; i--) {
          Frame frame1 = frame.vm.vmStack.get(i);
          if (isThrowable) {
              if (Instanceof.instanceof_(frame1.methodCode.className, "java/lang/Throwable",
frame)) {
                  continue;
              } else {
                  isThrowable = false;
              }
          }
          frames.add(frame1);
      }

      ObjectRef stackTraceElementsRef =
frame.heap.newArray(VariableType.STACK_TRACE_ELEMENT_ARRAY_TYPE, frames.size());
      frame.heap.putField(objectRef, "stackTraceElements", stackTraceElementsRef);
```

```
        int index = 0;
        for (Frame frame1 : frames) {
            ObjectRef declaringClass = frame.heap.newString(frame1.methodCode.className);
            ObjectRef methodName = frame.heap.newString(frame1.methodCode.name);
            ObjectRef fileName = frame.heap.newString(frame1.methodCode.classCode.sourceFile);
            int lineNumber = frame1.getLine();
            ObjectRef stackTraceElementRef =
frame.heap.runNewObjectWithArgs(VariableType.STACK_TRACE_ELEMENT_TYPE, null,
declaringClass, methodName, fileName, lineNumber);
            frame.heap.putArray(stackTraceElementsRef, index++, stackTraceElementRef);
        }

        Result result = NativeMethod.result(methodCode, objectRef, frame);
        return result;
    }

    public static final String getStackTraceDepth = TYPE + "." + "getStackTraceDepth" + "()I";

    /**
     * native
     *
     * @see Throwable#getStackTraceDepth()
     */
    private static Result getStackTraceDepth(MethodCode methodCode, MethodArgs methodArgs,
Frame frame) {
        ObjectRef objectRef = methodArgs.objectRef;
        ObjectRef stackTraceElementsRef = (ObjectRef) frame.heap.getField(objectRef,
"stackTraceElements");
        int depth = stackTraceElementsRef.getDimensions()[0];
        Result result = NativeMethod.result(methodCode, depth, frame);
        return result;
    }

    public static final String getStackTraceElement = TYPE + "." + "getStackTraceElement" +
"(I)Ljava/lang/StackTraceElement;";

    /**
     * native
     *
     * @see Throwable#getStackTraceElement(int)
     */
    private static Result getStackTraceElement(MethodCode methodCode, MethodArgs
```

```
methodArgs, Frame frame) {
        int index = (int) methodArgs.invokeArgs[0];
        ObjectRef objectRef = methodArgs.objectRef;
        ObjectRef stackTraceElementsRef = (ObjectRef) frame.heap.getField(objectRef,
"stackTraceElements");
        ObjectRef stackTraceElementRef = (ObjectRef)
frame.heap.getArray(stackTraceElementsRef, index);
        Result result = NativeMethod.result(methodCode, stackTraceElementRef, frame);
        return result;
    }

}

87:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\lang\reflect\NativeArray.java
 */
package io.nuls.contract.vm.natives.java.lang.reflect;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.natives.NativeMethod;

import java.lang.reflect.Array;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;

public class NativeArray {

    public static final String TYPE = "java/lang/reflect/Array";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case newArray:
                if (check) {
                    return SUCCESS;
                } else {
                    return newArray(methodCode, methodArgs, frame);
```

```
            }
        default:
            frame.nonsupportMethod(methodCode);
            return null;
    }
  }

  public static final String newArray = TYPE + "." + "newArray" +
"(Ljava/lang/Class;I)Ljava/lang/Object;";

  /**
   * native
   *
   * @see Array#newArray(Class, int)
   */
  private static Result newArray(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
      ObjectRef componentType = (ObjectRef) methodArgs.invokeArgs[0];
      int length = (int) methodArgs.invokeArgs[1];
      VariableType variableType = VariableType.valueOf("[" + componentType.getRef());
      ObjectRef array = frame.heap.newArray(variableType, length);
      Result result = NativeMethod.result(methodCode, array, frame);
      return result;
  }

}

88:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\java\sun\misc\NativeVM.java
 */
package io.nuls.contract.vm.natives.java.sun.misc;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.natives.NativeMethod;
import sun.misc.VM;

import static io.nuls.contract.vm.natives.NativeMethod.SUCCESS;
```

```
public class NativeVM {

    public static final String TYPE = "sun/misc/VM";

    public static Result nativeRun(MethodCode methodCode, MethodArgs methodArgs, Frame
frame, boolean check) {
        switch (methodCode.fullName) {
            case initialize:
                if (check) {
                    return SUCCESS;
                } else {
                    return initialize(methodCode, methodArgs, frame);
                }
            default:
                frame.nonsupportMethod(methodCode);
                return null;
        }
    }

    public static final String initialize = TYPE + "." + "initialize" + "()V";

    /**
     * native
     *
     * @see VM#initialize()
     */
    private static Result initialize(MethodCode methodCode, MethodArgs methodArgs, Frame
frame) {
        ObjectRef savedProps = (ObjectRef) frame.heap.getStatic(TYPE, "savedProps");
        ObjectRef key = frame.heap.newString("user.script");
        ObjectRef value = frame.heap.newString("");
        MethodCode methodCode1 = frame.methodArea.loadMethod("java/util/Properties", "put",
"(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;");
        frame.vm.run(methodCode1, new Object[]{savedProps, key, value}, false);
        Result result = NativeMethod.result(methodCode, null, frame);
        return result;
    }

}

89:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\natives\NativeMethod.java
```

```
 */
package io.nuls.contract.vm.natives;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.MethodArgs;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeAddress;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeBlock;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeMsg;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeUtils;
import io.nuls.contract.vm.natives.java.lang.*;
import io.nuls.contract.vm.natives.java.lang.reflect.NativeArray;
import io.nuls.contract.vm.natives.java.sun.misc.NativeVM;
import io.nuls.contract.vm.util.Log;

public class NativeMethod {

    private static final String registerNatives = "registerNatives";

    public static final Result SUCCESS = new Result();

    public static Result run(MethodCode methodCode, MethodArgs methodArgs, Frame frame,
boolean check) {

        Result result = null;

        switch (methodCode.className) {
            case NativeAbstractStringBuilder.TYPE:
                result = NativeAbstractStringBuilder.override(methodCode, methodArgs, frame, check);
                break;
            case NativeCharacter.TYPE:
                result = NativeCharacter.override(methodCode, methodArgs, frame, check);
                break;
            case NativeClass.TYPE:
                result = NativeClass.override(methodCode, methodArgs, frame, check);
                break;
            case NativeDouble.TYPE:
                result = NativeDouble.override(methodCode, methodArgs, frame, check);
                break;
```

```
    case NativeFloat.TYPE:
        result = NativeFloat.override(methodCode, methodArgs, frame, check);
        break;
    default:
        break;
}

if (result != null) {
    return result;
}

if (methodCode.instructions.size() > 0) {
    return null;
}

if (registerNatives.equals(methodCode.name)) {
    return new Result(methodCode.returnVariableType);
}

//Log.nativeMethod(methodCode);

switch (methodCode.className) {
    case NativeArray.TYPE:
        result = NativeArray.nativeRun(methodCode, methodArgs, frame, check);
        break;
    case NativeClass.TYPE:
        result = NativeClass.nativeRun(methodCode, methodArgs, frame, check);
        break;
    case NativeDouble.TYPE:
        result = NativeDouble.nativeRun(methodCode, methodArgs, frame, check);
        break;
    case NativeFloat.TYPE:
        result = NativeFloat.nativeRun(methodCode, methodArgs, frame, check);
        break;
    case NativeObject.TYPE:
        result = NativeObject.nativeRun(methodCode, methodArgs, frame, check);
        break;
    case NativeStrictMath.TYPE:
        result = NativeStrictMath.nativeRun(methodCode, methodArgs, frame, check);
        break;
    case NativeString.TYPE:
        result = NativeString.nativeRun(methodCode, methodArgs, frame, check);
```

```
            break;
        case NativeSystem.TYPE:
            result = NativeSystem.nativeRun(methodCode, methodArgs, frame, check);
            break;
        case NativeThrowable.TYPE:
            result = NativeThrowable.nativeRun(methodCode, methodArgs, frame, check);
            break;
        case NativeAddress.TYPE:
            result = NativeAddress.nativeRun(methodCode, methodArgs, frame, check);
            break;
        case NativeBlock.TYPE:
            result = NativeBlock.nativeRun(methodCode, methodArgs, frame, check);
            break;
        case NativeMsg.TYPE:
            result = NativeMsg.nativeRun(methodCode, methodArgs, frame, check);
            break;
        case NativeUtils.TYPE:
            result = NativeUtils.nativeRun(methodCode, methodArgs, frame, check);
            break;
        case NativeVM.TYPE:
            result = NativeVM.nativeRun(methodCode, methodArgs, frame, check);
            break;
        default:
            frame.nonsupportMethod(methodCode);
            break;
    }

    //Log.nativeMethodResult(result);

    return result;
}

public static Result run(MethodCode methodCode, MethodArgs methodArgs, Frame frame) {
    return run(methodCode, methodArgs, frame, false);
}

public static Result result(MethodCode methodCode, Object resultValue, Frame frame) {
    VariableType variableType = methodCode.returnVariableType;
    Result result = new Result(variableType);
    if (variableType.isNotVoid()) {
        result.value(resultValue);
        if (resultValue == null) {
```

```
                frame.operandStack.pushRef(null);
            } else if (variableType.isPrimitive()) {
                frame.operandStack.push(resultValue, variableType);
            } else if (resultValue instanceof ObjectRef) {
                frame.operandStack.pushRef((ObjectRef) resultValue);
            } else {
                throw new IllegalArgumentException("unknown result value");
            }
        }
        return result;
    }

}

90:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\ObjectRef.java
 */
package io.nuls.contract.vm;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.google.common.collect.BiMap;
import io.nuls.contract.vm.code.VariableType;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class ObjectRef {

    public static final Map<String, Integer> map = new HashMap<>();

    private final String ref;

    private final String desc;

    private final int[] dimensions;

    @JsonIgnore
    private final VariableType variableType;

    public ObjectRef(String ref, String desc, int... dimensions) {
        this.ref = ref;
```

```
        this.desc = desc;
        this.dimensions = dimensions;
        this.variableType = VariableType.valueOf(this.desc);
    }

    public ObjectRef(String str, BiMap<String, String> classNames) {
        String[] parts = str.split(",");
        int[] dimensions = new int[parts.length - 2];
        for (int i = 0; i < dimensions.length; i++) {
            int dimension = Integer.valueOf(parts[i + 2]);
            dimensions[i] = dimension;
        }
        this.ref = parts[0];
        String s = parts[1];
        if (classNames.containsKey(s)) {
            s = classNames.get(s);
        }
        this.desc = s;
        this.dimensions = dimensions;
        this.variableType = VariableType.valueOf(this.desc);
    }

    public String getEncoded(BiMap<String, String> classNames) {
        StringBuilder sb = new StringBuilder();
//        Integer i = map.get(desc);
//        if (i == null) {
//            i = 0;
//        }
//        map.put(desc, i + 1);
        String s = desc;
        if (classNames.inverse().containsKey(s)) {
            s = classNames.inverse().get(s);
        }
        sb.append(ref).append(",").append(s);
        for (int dimension : dimensions) {
            sb.append(",").append(dimension);
        }
        return sb.toString();
    }

    public boolean isArray() {
        return this.dimensions != null && this.dimensions.length > 0;
```

```
}

public String getRef() {
    return ref;
}

public String getDesc() {
    return desc;
}

public int[] getDimensions() {
    return dimensions;
}

public VariableType getVariableType() {
    return variableType;
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    ObjectRef objectRef = (ObjectRef) o;

    if (ref != null ? !ref.equals(objectRef.ref) : objectRef.ref != null) {
        return false;
    }
    if (desc != null ? !desc.equals(objectRef.desc) : objectRef.desc != null) {
        return false;
    }
    return Arrays.equals(dimensions, objectRef.dimensions);
}

@Override
public int hashCode() {
    int result = ref != null ? ref.hashCode() : 0;
    result = 31 * result + (desc != null ? desc.hashCode() : 0);
```

```
        result = 31 * result + Arrays.hashCode(dimensions);
        return result;
    }

    @Override
    public String toString() {
        return "ObjectRef{" +
                "ref=" + ref +
                ", desc=" + desc +
                ", dimensions=" + Arrays.toString(dimensions) +
                '}';
    }

}

91:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\OpCode.java
 */
package io.nuls.contract.vm;

public enum OpCode {

    NOP(0), // visitInsn
    ACONST_NULL(1), // -
    ICONST_M1(2), // -
    ICONST_0(3), // -
    ICONST_1(4), // -
    ICONST_2(5), // -
    ICONST_3(6), // -
    ICONST_4(7), // -
    ICONST_5(8), // -
    LCONST_0(9), // -
    LCONST_1(10), // -
    FCONST_0(11), // -
    FCONST_1(12), // -
    FCONST_2(13), // -
    DCONST_0(14), // -
    DCONST_1(15), // -
    BIPUSH(16), // visitIntInsn
    SIPUSH(17), // -
    LDC(18), // visitLdcInsn
    // LDC_W(19), // -
```

// LDC2_W(20), // -
ILOAD(21), // visitVarInsn
LLOAD(22), // -
FLOAD(23), // -
DLOAD(24), // -
ALOAD(25), // -
// ILOAD_0(26), // -
// ILOAD_1(27), // -
// ILOAD_2(28), // -
// ILOAD_3(29), // -
// LLOAD_0(30), // -
// LLOAD_1(31), // -
// LLOAD_2(32), // -
// LLOAD_3(33), // -
// FLOAD_0(34), // -
// FLOAD_1(35), // -
// FLOAD_2(36), // -
// FLOAD_3(37), // -
// DLOAD_0(38), // -
// DLOAD_1(39), // -
// DLOAD_2(40), // -
// DLOAD_3(41), // -
// ALOAD_0(42), // -
// ALOAD_1(43), // -
// ALOAD_2(44), // -
// ALOAD_3(45), // -
IALOAD(46), // visitInsn
LALOAD(47), // -
FALOAD(48), // -
DALOAD(49), // -
AALOAD(50), // -
BALOAD(51), // -
CALOAD(52), // -
SALOAD(53), // -
ISTORE(54), // visitVarInsn
LSTORE(55), // -
FSTORE(56), // -
DSTORE(57), // -
ASTORE(58), // -
// ISTORE_0(59), // -
// ISTORE_1(60), // -
// ISTORE_2(61), // -

// ISTORE_3(62), // -
// LSTORE_0(63), // -
// LSTORE_1(64), // -
// LSTORE_2(65), // -
// LSTORE_3(66), // -
// FSTORE_0(67), // -
// FSTORE_1(68), // -
// FSTORE_2(69), // -
// FSTORE_3(70), // -
// DSTORE_0(71), // -
// DSTORE_1(72), // -
// DSTORE_2(73), // -
// DSTORE_3(74), // -
// ASTORE_0(75), // -
// ASTORE_1(76), // -
// ASTORE_2(77), // -
// ASTORE_3(78), // -
IASTORE(79), // visitInsn
LASTORE(80), // -
FASTORE(81), // -
DASTORE(82), // -
AASTORE(83), // -
BASTORE(84), // -
CASTORE(85), // -
SASTORE(86), // -
POP(87), // -
POP2(88), // -
DUP(89), // -
DUP_X1(90), // -
DUP_X2(91), // -
DUP2(92), // -
DUP2_X1(93), // -
DUP2_X2(94), // -
SWAP(95), // -
IADD(96), // -
LADD(97), // -
FADD(98), // -
DADD(99), // -
ISUB(100), // -
LSUB(101), // -
FSUB(102), // -
DSUB(103), // -

IMUL(104), // -
LMUL(105), // -
FMUL(106), // -
DMUL(107), // -
IDIV(108), // -
LDIV(109), // -
FDIV(110), // -
DDIV(111), // -
IREM(112), // -
LREM(113), // -
FREM(114), // -
DREM(115), // -
INEG(116), // -
LNEG(117), // -
FNEG(118), // -
DNEG(119), // -
ISHL(120), // -
LSHL(121), // -
ISHR(122), // -
LSHR(123), // -
IUSHR(124), // -
LUSHR(125), // -
IAND(126), // -
LAND(127), // -
IOR(128), // -
LOR(129), // -
IXOR(130), // -
LXOR(131), // -
IINC(132), // visitIincInsn
I2L(133), // visitInsn
I2F(134), // -
I2D(135), // -
L2I(136), // -
L2F(137), // -
L2D(138), // -
F2I(139), // -
F2L(140), // -
F2D(141), // -
D2I(142), // -
D2L(143), // -
D2F(144), // -
I2B(145), // -

I2C(146), // -
I2S(147), // -
LCMP(148), // -
FCMPL(149), // -
FCMPG(150), // -
DCMPL(151), // -
DCMPG(152), // -
IFEQ(153), // visitJumpInsn
IFNE(154), // -
IFLT(155), // -
IFGE(156), // -
IFGT(157), // -
IFLE(158), // -
IF_ICMPEQ(159), // -
IF_ICMPNE(160), // -
IF_ICMPLT(161), // -
IF_ICMPGE(162), // -
IF_ICMPGT(163), // -
IF_ICMPLE(164), // -
IF_ACMPEQ(165), // -
IF_ACMPNE(166), // -
GOTO(167), // -
JSR(168), // -
RET(169), // visitVarInsn
TABLESWITCH(170), // visiTableSwitchInsn
LOOKUPSWITCH(171), // visitLookupSwitch
IRETURN(172), // visitInsn
LRETURN(173), // -
FRETURN(174), // -
DRETURN(175), // -
ARETURN(176), // -
RETURN(177), // -
GETSTATIC(178), // visitFieldInsn
PUTSTATIC(179), // -
GETFIELD(180), // -
PUTFIELD(181), // -
INVOKEVIRTUAL(182), // visitMethodInsn
INVOKESPECIAL(183), // -
INVOKESTATIC(184), // -
INVOKEINTERFACE(185), // -
INVOKEDYNAMIC(186), // visitInvokeDynamicInsn
NEW(187), // visitTypeInsn

```
NEWARRAY(188), // visitIntInsn
ANEWARRAY(189), // visitTypeInsn
ARRAYLENGTH(190), // visitInsn
ATHROW(191), // -
CHECKCAST(192), // visitTypeInsn
INSTANCEOF(193), // -
MONITORENTER(194), // visitInsn
MONITOREXIT(195), // -
// WIDE(196), // NOT VISITED
MULTIANEWARRAY(197), // visitMultiANewArrayInsn
IFNULL(198), // visitJumpInsn
IFNONNULL(199), // -
// GOTO_W(200), // -
// JSR_W(201), // -

;

private static final int SIZE = 202;

private int opcode;

OpCode(int opcode) {
    this.opcode = opcode;
}

public int getOpcode() {
    return opcode;
}

public static final OpCode[] OPCODES;

static {
    OPCODES = new OpCode[SIZE];
    for (OpCode opCode : values()) {
        OPCODES[opCode.getOpcode()] = opCode;
    }
}

public static OpCode valueOf(int opcode) {
    if (opcode < 0 || opcode >= SIZE) {
        return null;
    } else {
```

```
            return OPCODES[opcode];
        }
    }

}

92:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\OperandStack.java
 */
package io.nuls.contract.vm;

import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.code.VariableType;

import java.util.Stack;

public class OperandStack extends Stack {

    private final int maxStack;

    public OperandStack(int maxStack) {
        super();
        this.maxStack = maxStack;
    }

    @Override
    public Object push(Object value) {
        return super.push(value);
    }

    public Object push(Object value, VariableType variableType) {
        if (variableType.isPrimitive()) {
            switch (variableType.getType()) {
                case Descriptors.INT:
                    value = pushInt((int) value);
                    break;
                case Descriptors.LONG:
                    value = pushLong((long) value);
                    break;
                case Descriptors.FLOAT:
                    value = pushFloat((float) value);
                    break;
```

```
            case Descriptors.DOUBLE:
                value = pushDouble((double) value);
                break;
            case Descriptors.BOOLEAN:
                value = pushBoolean((boolean) value);
                break;
            case Descriptors.BYTE:
                value = pushByte((byte) value);
                break;
            case Descriptors.CHAR:
                value = pushChar((char) value);
                break;
            case Descriptors.SHORT:
                value = pushShort((short) value);
                break;
            default:
                value = push(value);
                break;
        }
    } else {
        value = push(value);
    }
    return value;
}

@Override
public synchronized Object pop() {
    return super.pop();
}

public int pushInt(int value) {
    push(value);
    return value;
}

public int popInt() {
    return (int) pop();
}

public long pushLong(long value) {
    push(value);
    push(null);
```

```
    return value;
}

public long popLong() {
    pop();
    return (long) pop();
}

public float pushFloat(float value) {
    push(value);
    return value;
}

public float popFloat() {
    return (float) pop();
}

public double pushDouble(double value) {
    push(value);
    push(null);
    return value;
}

public double popDouble() {
    pop();
    return (double) pop();
}

public int pushBoolean(boolean value) {
    return pushInt(value ? 1 : 0);
}

public boolean popBoolean() {
    return popInt() == 1 ? true : false;
}

public int pushByte(byte value) {
    return pushInt(value);
}

public byte popByte() {
    return (byte) popInt();
```

```
    }

    public int pushChar(char value) {
        return pushInt(value);
    }

    public char popChar() {
        return (char) popInt();
    }

    public int pushShort(short value) {
        return pushInt(value);
    }

    public short popShort() {
        return (short) popInt();
    }

    public ObjectRef pushRef(ObjectRef ref) {
        push(ref);
        return ref;
    }

    public ObjectRef popRef() {
        return (ObjectRef) pop();
    }

}

93:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\KeyValueSource.java
 */
package io.nuls.contract.vm.program.impl;

import com.google.common.cache.Cache;
import com.google.common.cache.CacheBuilder;
import io.nuls.db.service.DBService;
import org.apache.commons.lang3.ArrayUtils;
import org.ethereum.datasource.Source;
import org.ethereum.db.ByteArrayWrapper;

import java.util.concurrent.TimeUnit;
```

```
public class KeyValueSource implements Source<byte[], byte[]> {

    public static final String AREA = "contract";

    private DBService dbService;

    private final Cache<ByteArrayWrapper, byte[]> cache;

    public KeyValueSource(DBService dbService) {
        this.dbService = dbService;
        String[] areas = dbService.listArea();
        if (!ArrayUtils.contains(areas, AREA)) {
            dbService.createArea(AREA);
        }
        this.cache = CacheBuilder.newBuilder()
                .initialCapacity(10240)
                .maximumSize(102400)
                .expireAfterAccess(10, TimeUnit.MINUTES)
                .build();
    }

    @Override
    public void put(byte[] key, byte[] val) {
        cache.put(new ByteArrayWrapper(key), val);
        dbService.put(AREA, key, val);
    }

    @Override
    public byte[] get(byte[] key) {
        byte[] bytes = cache.getIfPresent(new ByteArrayWrapper(key));
        if (bytes == null) {
            bytes = dbService.get(AREA, key);
        }
        return bytes;
    }

    @Override
    public void delete(byte[] key) {
    }

    @Override
```

```
    public boolean flush() {
        return true;
    }

}

94:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramChecker.java
 */
package io.nuls.contract.vm.program.impl;

import com.google.common.base.Joiner;
import io.nuls.contract.vm.OpCode;
import io.nuls.contract.vm.code.*;
import org.apache.commons.collections4.CollectionUtils;
import org.objectweb.asm.tree.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.*;
import java.util.stream.Collectors;

public class ProgramChecker {

    private static final Logger log = LoggerFactory.getLogger(ProgramExecutorImpl.class);

    public static void check(Map<String, ClassCode> classCodes) {
        checkJdkVersion(classCodes);
        checkContractNum(classCodes);
        //checkStaticField(classCodes);
        checkClass(classCodes);
        checkContractMethodArgs(classCodes);
        checkMethod(classCodes);
        checkOpCode(classCodes);
    }

    public static void checkJdkVersion(Map<String, ClassCode> classCodes) {
        for (ClassCode classCode : classCodes.values()) {
            if (!classCode.isV1_6 && !classCode.isV1_8) {
                throw new RuntimeException("class version must be 1.6 or 1.8");
            }
        }
```

```
    }

    public static void checkContractNum(Map<String, ClassCode> classCodes) {
        List<ClassCode> contractClassCodes = classCodes.values().stream()
                .filter(classCode ->
classCode.interfaces.contains(ProgramConstants.CONTRACT_INTERFACE_NAME))
                .collect(Collectors.toList());
        int contractCount = contractClassCodes.size();
        if (contractCount != 1) {
            throw new RuntimeException(String.format("find %s contracts", contractCount));
        }
    }

    public static void checkContractMethodArgs(Map<String, ClassCode> classCodes) {
        final List<MethodCode> list = ProgramExecutorImpl.getProgramMethodCodes(classCodes);
        for (MethodCode methodCode : list) {
            final List<VariableType> variableTypes = methodCode.argsVariableType;
            for (VariableType variableType : variableTypes) {
                if (variableType.isPrimitive()) {
                    //
                } else if (variableType.isArray()) {
                    if (variableType.getDimensions() > 1) {
                        throw new RuntimeException(String.format("only one-dimensional array can be
used in method %s.%s", methodCode.className, methodCode.name));
                    } else if (!(variableType.isPrimitiveType() ||
variableType.getComponentType().isStringType() ||
variableType.getComponentType().isWrapperType())) {
                        throw new RuntimeException(String.format("only primitive type array and string
array can be used in method %s.%s", methodCode.className, methodCode.name));
                    } else {
                        //
                    }
                } else {
                    if (!hasConstructor(variableType, classCodes)) {
                        throw new RuntimeException(String.format("%s can't be used in method %s.%s",
variableType.getType(), methodCode.className, methodCode.name));
                    } else {
                        //
                    }
                }
            }
        }
```

```
    }

    public static void checkStaticField(Map<String, ClassCode> classCodes) {
        List<FieldCode> fieldCodes = new ArrayList<>();
        classCodes.values().stream().forEach(classCode -> {
            List<FieldCode> list = classCode.fields.values().stream().filter(fieldCode ->
fieldCode.isStatic).collect(Collectors.toList());
            fieldCodes.addAll(list);
        });
        if (fieldCodes.size() > 0) {
            throw new RuntimeException(String.format("find %s static fields", fieldCodes.size()));
        }
    }

    public static void checkClass(Map<String, ClassCode> classCodes) {
        Set<String> allClass = allClass(classCodes);
        Set<String> classCodeNames = classCodes.values().stream().map(classCode ->
classCode.name).collect(Collectors.toSet());
        Collection<String> classes = CollectionUtils.removeAll(allClass, classCodeNames);
        Collection<String> classes1 = CollectionUtils.removeAll(classes,
Arrays.asList(ProgramConstants.SDK_CLASS_NAMES));
        Collection<String> classes2 = CollectionUtils.removeAll(classes1,
Arrays.asList(ProgramConstants.CONTRACT_USED_CLASS_NAMES));
        Collection<String> classes3 = CollectionUtils.removeAll(classes2,
Arrays.asList(ProgramConstants.CONTRACT_LAZY_USED_CLASS_NAMES));
        if (classes3.size() > 0) {
            throw new RuntimeException(String.format("can't use classes: %s", Joiner.on(",
").join(classes3)));
        }
    }

    public static void checkMethod(Map<String, ClassCode> classCodes) {
        Map<String, Object> methodCodes = new HashMap(1024);
        for (ClassCode classCode : classCodes.values()) {
            for (MethodCode methodCode : classCode.methods) {
                Object o = methodCodes.get(methodCode.fullName);
                if (o == null) {
                    o = isSupportMethod(methodCode, methodCodes, classCodes);
                    methodCodes.put(methodCode.fullName, o);
                }
                if (!Boolean.TRUE.equals(o)) {
                    if (o != null) {
```

```
                    MethodInsnNode methodInsnNode = (MethodInsnNode) o;
                    throw new RuntimeException(String.format("can't use method: %s.%s%s",
methodInsnNode.owner, methodInsnNode.name, methodInsnNode.desc));
                } else {
                    throw new RuntimeException(String.format("can't use method: %s.%s%s",
methodCode.className, methodCode.name, methodCode.desc));
                }
            }
        }
    }
}

public static void checkOpCode(Map<String, ClassCode> classCodes) {
    for (ClassCode classCode : classCodes.values()) {
        for (MethodCode methodCode : classCode.methods) {
            checkOpCode(methodCode);
        }
    }
}

public static void checkOpCode(MethodCode methodCode) {
    ListIterator<AbstractInsnNode> listIterator = methodCode.instructions.iterator();
    while (listIterator.hasNext()) {
        AbstractInsnNode abstractInsnNode = listIterator.next();
        if (abstractInsnNode != null && abstractInsnNode.getOpcode() > 0) {
            OpCode opCode = OpCode.valueOf(abstractInsnNode.getOpcode());
            boolean nonsupport = false;
            if (opCode == null) {
                nonsupport = true;
            } else {
                switch (opCode) {
                    case JSR:
                    case RET:
                    case INVOKEDYNAMIC:
                    case MONITORENTER:
                    case MONITOREXIT:
                        nonsupport = true;
                        break;
                    default:
                        break;
                }
            }
```

```
            if (nonsupport) {
                int line = getLine(abstractInsnNode);
                throw new RuntimeException(String.format("nonsupport opcode: class(%s), line(%d)",
methodCode.className, line));
            }
        }
    }
}

public static Set<String> allClass(Map<String, ClassCode> classCodes) {
    Set<String> set = new HashSet<>(100);
    for (ClassCode classCode : classCodes.values()) {
        set.add(classCode.name);
        set.add(classCode.superName);
        set.addAll(classCode.interfaces);
        for (InnerClassNode innerClassNode : classCode.innerClasses) {
            set.add(innerClassNode.name);
        }
        for (FieldCode fieldCode : classCode.fields.values()) {
            set.add(fieldCode.desc);
        }
        for (MethodCode methodCode : classCode.methods) {
            set.addAll(allClass(methodCode));
        }
    }

    Set<String> classes = new HashSet<>(set.size() * 5);
    for (String s : set) {
        if (s == null) {
            continue;
        }
        if (s.contains("$")) {
            //continue;
        }
        if (s.contains("(")) {
            List<VariableType> list = VariableType.parseAll(s);
            for (VariableType variableType : list) {
                if (!variableType.isPrimitiveType() && variableType.isNotVoid()) {
                    classes.add(variableType.getType());
                }
            }
        } else {
```

```
                VariableType variableType = VariableType.valueOf(s);
                if (!variableType.isPrimitiveType() && variableType.isNotVoid()) {
                    classes.add(variableType.getType());
                }
            }
        }
        return classes;
    }

    public static Set<String> allClass(MethodCode methodCode) {
        Set<String> set = new HashSet<>();
        set.add(methodCode.returnVariableType.getType());
        for (VariableType variableType : methodCode.argsVariableType) {
            set.add(variableType.getType());
        }
        ListIterator<AbstractInsnNode> listIterator = methodCode.instructions.iterator();
        while (listIterator.hasNext()) {
            AbstractInsnNode abstractInsnNode = listIterator.next();
            if (abstractInsnNode instanceof MultiANewArrayInsnNode) {
                MultiANewArrayInsnNode insnNode = (MultiANewArrayInsnNode) abstractInsnNode;
                set.add(insnNode.desc);
            } else if (abstractInsnNode instanceof MethodInsnNode) {
                MethodInsnNode insnNode = (MethodInsnNode) abstractInsnNode;
                set.add(insnNode.owner);
                set.add(insnNode.desc);
            } else if (abstractInsnNode instanceof TypeInsnNode) {
                TypeInsnNode insnNode = (TypeInsnNode) abstractInsnNode;
                set.add(insnNode.desc);
            } else if (abstractInsnNode instanceof FieldInsnNode) {
                FieldInsnNode insnNode = (FieldInsnNode) abstractInsnNode;
                set.add(insnNode.owner);
                set.add(insnNode.desc);
            }
        }
        return set;
    }

    public static Set<String> notSupportMethods = new TreeSet<>();

    public static Object isSupportMethod(MethodCode methodCode, Map<String, Object>
methodCodes, Map<String, ClassCode> classCodeMap) {
        if (!methodCodes.containsKey(methodCode.fullName)) {
```

```
            methodCodes.put(methodCode.fullName, null);
//            if (NativeMethod.isSupport(methodCode)) {
//                return Boolean.TRUE;
//            }
//            String methodFullName = String.format("%s.%s%s", methodCode.className,
methodCode.name, methodCode.desc);
//            if (methodCode.className.startsWith("sun/")) {
//                notSupportMethods.add(methodFullName);
//                log.warn("not support sun method " + methodFullName);
//                return null;
//            }
//            if (methodCode.getInstructions().size() <= 0) {
//                if (methodCode.isNative()) {
//                    if (ArrayUtils.contains(NativeMethod.SUPPORT_CLASSES,
methodCode.className)) {
//                        return Boolean.TRUE;
//                    } else {
//                        notSupportMethods.add(methodFullName);
//                        log.warn("not support native method " + methodFullName);
//                        return null;
//                    }
//                } else if (methodCode.classCode.isInterface() || methodCode.isAbstract()) {
//                    return Boolean.TRUE;
//                } else {
//                    notSupportMethods.add(methodFullName);
//                    log.warn("not support native method " + methodFullName);
//                    return null;
//                }
//            }
            ListIterator<AbstractInsnNode> listIterator = methodCode.instructions.iterator();
            while (listIterator.hasNext()) {
                AbstractInsnNode abstractInsnNode = listIterator.next();
                if (!(abstractInsnNode instanceof MethodInsnNode)) {
                    continue;
                }
                MethodInsnNode methodInsnNode = (MethodInsnNode) abstractInsnNode;
                VariableType variableType = VariableType.valueOf(methodInsnNode.owner);
                if (variableType.isPrimitiveType()) {
                    continue;
                }
                ClassCode classCode = getClassCode(variableType.getType(), classCodeMap);
                if (classCode == null) {
```

```
                log.warn("can't find class " + methodInsnNode.owner);
                return methodInsnNode;
            }
            MethodCode methodCode1 = getMethodCode(classCode, methodInsnNode.name,
methodInsnNode.desc, classCodeMap);
            if (methodCode1 == null) {
                log.warn(String.format("can't find method %s.%s%s", methodInsnNode.owner,
methodInsnNode.name, methodInsnNode.desc));
                return methodInsnNode;
            }
            Object o = isSupportMethod(methodCode1, methodCodes, classCodeMap);
            if (!Boolean.TRUE.equals(o)) {
                log.warn(String.format("not support method %s.%s%s", methodInsnNode.owner,
methodInsnNode.name, methodInsnNode.desc));
                return methodInsnNode;
            }
        }
    }
    return Boolean.TRUE;
  }

  public static MethodCode getMethodCode(ClassCode classCode, String methodName, String
methodDesc, Map<String, ClassCode> classCodeMap) {
    MethodCode methodCode = classCode.getMethodCode(methodName, methodDesc);
    if (methodCode == null && classCode.superName != null) {
        ClassCode superClassCode = getClassCode(classCode.superName, classCodeMap);
        if (superClassCode != null) {
            methodCode = getMethodCode(superClassCode, methodName, methodDesc,
classCodeMap);
        }
    }
    if (methodCode == null) {
        for (String interfaceName : classCode.interfaces) {
            ClassCode interfaceClassCode = getClassCode(interfaceName, classCodeMap);
            methodCode = getMethodCode(interfaceClassCode, methodName, methodDesc,
classCodeMap);
            if (methodCode != null) {
                break;
            }
        }
    }
    return methodCode;
```

```
    }

    public static int getLine(AbstractInsnNode abstractInsnNode) {
        while (!(abstractInsnNode instanceof LineNumberNode)) {
            abstractInsnNode = abstractInsnNode.getPrevious();
        }
        return ((LineNumberNode) abstractInsnNode).line;
    }

    public static boolean hasConstructor(VariableType variableType, Map<String, ClassCode>
classCodeMap) {
        ClassCode classCode = getClassCode(variableType.getType(), classCodeMap);
        if (classCode != null) {
            MethodCode methodCode = classCode.getMethodCode("<init>", "(Ljava/lang/String;)V");
            if (methodCode != null && methodCode.isPublic) {
                return true;
            }
        }
        return false;
    }

    private static ClassCode getClassCode(String className, Map<String, ClassCode>
classCodeMap) {
        ClassCode classCode = classCodeMap.get(className);
        if (classCode == null) {
            classCode = ClassCodeLoader.getFromResource(className);
        }
        return classCode;
    }

}

95:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramConstants.java
 */
package io.nuls.contract.vm.program.impl;

import io.nuls.contract.sdk.*;
import io.nuls.contract.sdk.annotation.Payable;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;
```

```
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.*;

public class ProgramConstants {

    public static final String CONTRACT_INTERFACE_NAME =
classNameReplace(Contract.class.getName());

    public static final String EVENT_INTERFACE_NAME =
classNameReplace(Event.class.getName());

    public static final Class[] SDK_CLASSES = new Class[]{
            Address.class,
            Block.class,
            BlockHeader.class,
            Contract.class,
            Event.class,
            Msg.class,
            Utils.class,
            View.class,
            Required.class,
            Payable.class,
    };

    public static final Class[] CONTRACT_USED_CLASSES = new Class[]{
            Boolean.class,
            Byte.class,
            Short.class,
            Character.class,
            Integer.class,
            Long.class,
            Float.class,
            Double.class,
            String.class,
            StringBuilder.class,
            BigInteger.class,
            BigDecimal.class,
            List.class,
            ArrayList.class,
            Map.class,
            Map.Entry.class,
```

```
        HashMap.class,
        LinkedHashMap.class,
        Set.class,
        HashSet.class,
        Exception.class,
        RuntimeException.class,
    };

    public static final Class[] CONTRACT_LAZY_USED_CLASSES = new Class[]{
        Object.class,
        Class.class,
        Set.class,
        Iterator.class,
    };

    public static final Class[] VM_INIT_CLASSES = new Class[]{
        Object.class,
        Class.class,
        StrictMath.class,
        RuntimeException.class,
        ArrayIndexOutOfBoundsException.class,
        OutOfMemoryError.class,
        Collections.class,
        HashSet.class,
        StackOverflowError.class,
        NullPointerException.class,
        NegativeArraySizeException.class,
        ClassCastException.class,
        StackOverflowError.class,
    };

    public static final String[] SDK_CLASS_NAMES = new String[SDK_CLASSES.length];

    public static final String[] CONTRACT_USED_CLASS_NAMES = new
String[CONTRACT_USED_CLASSES.length];

    public static final String[] CONTRACT_LAZY_USED_CLASS_NAMES = new
String[CONTRACT_LAZY_USED_CLASSES.length];

    public static final String[] VM_INIT_CLASS_NAMES = new String[VM_INIT_CLASSES.length +
1];
```

```
    static {
        for (int i = 0; i < SDK_CLASSES.length; i++) {
            SDK_CLASS_NAMES[i] = classNameReplace(SDK_CLASSES[i].getName());
        }
        for (int i = 0; i < CONTRACT_USED_CLASSES.length; i++) {
            CONTRACT_USED_CLASS_NAMES[i] =
classNameReplace(CONTRACT_USED_CLASSES[i].getName());
        }
        for (int i = 0; i < CONTRACT_LAZY_USED_CLASSES.length; i++) {
            CONTRACT_LAZY_USED_CLASS_NAMES[i] =
classNameReplace(CONTRACT_LAZY_USED_CLASSES[i].getName());
        }
        int length = VM_INIT_CLASSES.length;
        for (int i = 0; i < length; i++) {
            VM_INIT_CLASS_NAMES[i] = classNameReplace(VM_INIT_CLASSES[i].getName());
        }
        VM_INIT_CLASS_NAMES[length] = "java/lang/CharacterDataLatin1";
    }

    public static String classNameReplace(String s) {
        return s.replace('.', '/');
    }

}

96:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramContext.java
 */
package io.nuls.contract.vm.program.impl;

import io.nuls.contract.vm.ObjectRef;

public class ProgramContext {

    private ObjectRef address;

    private ObjectRef sender;

    //private ObjectRef balance;

    private long gasPrice;
```

```
private long gas;

//private long gasLimit;

private ObjectRef value;

private long number;

//private long difficulty;

//private ObjectRef data;

private boolean estimateGas;

public ObjectRef getAddress() {
    return address;
}

public void setAddress(ObjectRef address) {
    this.address = address;
}

public ObjectRef getSender() {
    return sender;
}

public void setSender(ObjectRef sender) {
    this.sender = sender;
}

public long getGasPrice() {
    return gasPrice;
}

public void setGasPrice(long gasPrice) {
    this.gasPrice = gasPrice;
}

public long getGas() {
    return gas;
}
```

```
    public void setGas(long gas) {
        this.gas = gas;
    }

    public ObjectRef getValue() {
        return value;
    }

    public void setValue(ObjectRef value) {
        this.value = value;
    }

    public long getNumber() {
        return number;
    }

    public void setNumber(long number) {
        this.number = number;
    }

    public boolean isEstimateGas() {
        return estimateGas;
    }

    public void setEstimateGas(boolean estimateGas) {
        this.estimateGas = estimateGas;
    }

}

97:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramDescriptors.java
 */
package io.nuls.contract.vm.program.impl;

import com.google.common.collect.BiMap;
import com.google.common.collect.HashBiMap;
import io.nuls.contract.vm.code.Descriptors;
import io.nuls.contract.vm.code.VariableType;
import org.apache.commons.lang3.StringUtils;

import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;

public class ProgramDescriptors {

    public static final BiMap<String, String> DESCRIPTORS;

    static {
        DESCRIPTORS = HashBiMap.create();
        DESCRIPTORS.put("Boolean", "Ljava/lang/Boolean;");
        DESCRIPTORS.put("Byte", "Ljava/lang/Byte;");
        DESCRIPTORS.put("Short", "Ljava/lang/Short;");
        DESCRIPTORS.put("Character", "Ljava/lang/Character;");
        DESCRIPTORS.put("Integer", "Ljava/lang/Integer;");
        DESCRIPTORS.put("Long", "Ljava/lang/Long;");
        DESCRIPTORS.put("Float", "Ljava/lang/Float;");
        DESCRIPTORS.put("Double", "Ljava/lang/Double;");
        DESCRIPTORS.put("String", "Ljava/lang/String;");
        DESCRIPTORS.put("BigInteger", "Ljava/math/BigInteger;");
        DESCRIPTORS.put("Address", "Lio/nuls/contract/sdk/Address;");
    }

    private static final Pattern PATTERN = Pattern.compile("^\\((.*)\\) return (.+)$");

    public static String getNormalDesc(VariableType variableType) {
        String desc = variableType.getDesc().replace("[", "");
        if (Descriptors.DESCRIPTORS.inverse().containsKey(desc)) {
            desc = Descriptors.DESCRIPTORS.inverse().get(desc);
        } else {
            if (ProgramDescriptors.DESCRIPTORS.inverse().containsKey(desc)) {
                desc = ProgramDescriptors.DESCRIPTORS.inverse().get(desc);
            }
        }
        if (variableType.isArray()) {
            for (int i = 0; i < variableType.getDimensions(); i++) {
                desc += "[]";
            }
        }
        return desc;
    }

    public static String parseDesc(String desc) {
        if (desc == null) {
```

```
            return null;
        }

        desc = desc.trim();
        StringBuilder sb = new StringBuilder();

        Matcher matcher = PATTERN.matcher(desc);
        if (matcher.matches()) {
            sb.append("(");
            String arg = matcher.group(1);
            if (StringUtils.isNotEmpty(arg)) {
                String[] args = arg.split(", ");
                for (String s : args) {
                    sb.append(getDesc(s));
                }
            }
            sb.append(")");
            String returnArg = matcher.group(2);
            sb.append(getDesc(returnArg));
        } else {
            sb.append(desc);
        }

        return sb.toString();
    }

    private static String getDesc(String desc) {
        int dimensions = StringUtils.countMatches(desc, "[]");
        desc = desc.replace("[]", "");
        String[] parts = desc.split(" ");
        desc = parts[0];
        if (Descriptors.DESCRIPTORS.containsKey(desc)) {
            desc = Descriptors.DESCRIPTORS.get(desc);
        } else {
            if (ProgramDescriptors.DESCRIPTORS.containsKey(desc)) {
                desc = ProgramDescriptors.DESCRIPTORS.get(desc);
            }
        }
        for (int i = 0; i < dimensions; i++) {
            desc = "[" + desc;
        }
        return desc;
```

```
    }

}

98:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramExecutorImpl.java
 */
package io.nuls.contract.vm.program.impl;

import io.nuls.contract.entity.BlockHeaderDto;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.ObjectRef;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.VM;
import io.nuls.contract.vm.VMFactory;
import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.ClassCodeLoader;
import io.nuls.contract.vm.code.ClassCodes;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.exception.ErrorException;
import io.nuls.contract.vm.exception.RevertException;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeAddress;
import io.nuls.contract.vm.program.*;
import io.nuls.contract.vm.util.Constants;
import io.nuls.db.service.DBService;
import org.apache.commons.lang3.StringUtils;
import org.ethereum.config.CommonConfig;
import org.ethereum.config.DefaultConfig;
import org.ethereum.config.SystemProperties;
import org.ethereum.core.AccountState;
import org.ethereum.core.Block;
import org.ethereum.core.Repository;
import org.ethereum.datasource.Source;
import org.ethereum.datasource.leveldb.LevelDbDataSource;
import org.ethereum.db.ByteArrayWrapper;
import org.ethereum.db.RepositoryRoot;
import org.ethereum.db.StateSource;
import org.ethereum.util.FastByteComparisons;
import org.ethereum.vm.DataWord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.spongycastle.util.encoders.Hex;
```

```
import java.math.BigInteger;
import java.util.*;
import java.util.stream.Collectors;

public class ProgramExecutorImpl implements ProgramExecutor {

    private static final Logger log = LoggerFactory.getLogger(ProgramExecutorImpl.class);

    private final ProgramExecutorImpl parent;

    private final VMContext vmContext;

    private final Source<byte[], byte[]> source;

    private final Repository repository;

    private final byte[] prevStateRoot;

    private final long beginTime;

    private final Map<ByteArrayWrapper, ProgramAccount> accounts = new HashMap<>();

    private long blockNumber;

    private long currentTime;

    private boolean revert;

    private Thread thread;

    public ProgramExecutorImpl(VMContext vmContext, DBService dbService) {
        this(vmContext, stateSource(dbService), null, null, null);
    }

    private ProgramExecutorImpl(VMContext vmContext, Source<byte[], byte[]> source, Repository
repository, byte[] prevStateRoot, Thread thread) {
        this.parent = this;
        this.vmContext = vmContext;
        this.source = source;
        this.repository = repository;
        this.prevStateRoot = prevStateRoot;
```

```
        this.beginTime = this.currentTime = System.currentTimeMillis();
        this.thread = thread;
    }

    @Override
    public ProgramExecutor begin(byte[] prevStateRoot) {
        if (log.isDebugEnabled()) {
            log.debug("begin vm root: {}", Hex.toHexString(prevStateRoot));
        }
        Repository repository = new RepositoryRoot(source, prevStateRoot);
        return new ProgramExecutorImpl(vmContext, source, repository, prevStateRoot,
Thread.currentThread());
    }

    @Override
    public ProgramExecutor startTracking() {
        checkThread();
        if (log.isDebugEnabled()) {
            log.debug("startTracking");
        }
        Repository track = repository.startTracking();
        return new ProgramExecutorImpl(vmContext, source, track, null, thread);
    }

    @Override
    public void commit() {
        checkThread();
        if (!revert) {
            repository.commit();
            if (prevStateRoot == null) {
                if (parent.blockNumber == 0) {
                    parent.blockNumber = blockNumber;
                }
                if (parent.blockNumber != blockNumber) {
                    throw new RuntimeException("must use the same block number");
                }
            } else {
                if (vmContext != null) {
                    BlockHeaderDto blockHeaderDto;
                    try {
                        blockHeaderDto = vmContext.getBlockHeader(blockNumber);
                    } catch (Exception e) {
```

```
                    throw new RuntimeException(e);
                }
                byte[] parentHash = Hex.decode(blockHeaderDto.getPreHash());
                byte[] hash = Hex.decode(blockHeaderDto.getHash());
                Block block = new Block(parentHash, hash, blockNumber);
                DefaultConfig.getDefault().blockStore().saveBlock(block, BigInteger.ONE, true);
                DefaultConfig.getDefault().pruneManager().blockCommitted(block.getHeader());
            }
            CommonConfig.getDefault().dbFlushManager().flush();
        }
        logTime("commit");
    }
}

@Override
public byte[] getRoot() {
    checkThread();
    byte[] root;
    if (!revert) {
        root = repository.getRoot();
    } else {
        root = this.prevStateRoot;
    }
    if (log.isDebugEnabled()) {
        log.debug("end vm root: {}, runtime: {}", Hex.toHexString(root), System.currentTimeMillis()
- beginTime);
    }
    return root;
}

@Override
public ProgramResult create(ProgramCreate programCreate) {
    checkThread();
    ProgramInvoke programInvoke = new ProgramInvoke();
    programInvoke.setContractAddress(programCreate.getContractAddress());
    programInvoke.setAddress(NativeAddress.toString(programInvoke.getContractAddress()));
    programInvoke.setSender(programCreate.getSender());
    programInvoke.setPrice(programCreate.getPrice());
    programInvoke.setGasLimit(programCreate.getGasLimit());
    programInvoke.setValue(programCreate.getValue() != null ? programCreate.getValue() :
BigInteger.ZERO);
    programInvoke.setNumber(programCreate.getNumber());
```

```
        programInvoke.setData(programCreate.getContractCode());
        programInvoke.setMethodName("<init>");
        programInvoke.setArgs(programCreate.getArgs() != null ? programCreate.getArgs() : new
String[0][0]);
        programInvoke.setEstimateGas(programCreate.isEstimateGas());
        programInvoke.setCreate(true);
        programInvoke.setInternalCall(false);
        return execute(programInvoke);
    }

    @Override
    public ProgramResult call(ProgramCall programCall) {
        checkThread();
        ProgramInvoke programInvoke = new ProgramInvoke();
        programInvoke.setContractAddress(programCall.getContractAddress());
        programInvoke.setAddress(NativeAddress.toString(programInvoke.getContractAddress()));
        programInvoke.setSender(programCall.getSender());
        programInvoke.setPrice(programCall.getPrice());
        programInvoke.setGasLimit(programCall.getGasLimit());
        programInvoke.setValue(programCall.getValue() != null ? programCall.getValue() :
BigInteger.ZERO);
        programInvoke.setNumber(programCall.getNumber());
        programInvoke.setMethodName(programCall.getMethodName());
        programInvoke.setMethodDesc(programCall.getMethodDesc());
        programInvoke.setArgs(programCall.getArgs() != null ? programCall.getArgs() : new
String[0][0]);
        programInvoke.setEstimateGas(programCall.isEstimateGas());
        programInvoke.setCreate(false);
        programInvoke.setInternalCall(programCall.isInternalCall());
        return execute(programInvoke);
    }

    private ProgramResult execute(ProgramInvoke programInvoke) {
        if (programInvoke.getPrice() < 1) {
            return revert("gas price must be greater than zero");
        }
        if (programInvoke.getGasLimit() < 1) {
            return revert("gas must be greater than zero");
        }
        if (programInvoke.getGasLimit() > VM.MAX_GAS) {
            return revert("gas must be less than " + VM.MAX_GAS);
        }
```

```
        if (programInvoke.getValue().compareTo(BigInteger.ZERO) < 0) {
            return revert("value can't be less than zero");
        }
        blockNumber = programInvoke.getNumber();

        logTime("start");

        try {
            Map<String, ClassCode> classCodes;
            if (programInvoke.isCreate()) {
                if (programInvoke.getData() == null) {
                    return revert("contract code can't be null");
                }
                classCodes = ClassCodeLoader.loadJarCache(programInvoke.getData());
                logTime("load new code");
                ProgramChecker.check(classCodes);
                logTime("check code");
                AccountState accountState =
repository.getAccountState(programInvoke.getContractAddress());
                if (accountState != null) {
                    return revert(String.format("contract[%s] already exists",
programInvoke.getAddress()));
                }
                accountState = repository.createAccount(programInvoke.getContractAddress(),
programInvoke.getSender());
                logTime("new account state");
                repository.saveCode(programInvoke.getContractAddress(), programInvoke.getData());
                logTime("save code");
            } else {
                if ("<init>".equals(programInvoke.getMethodName())) {
                    return revert("can't invoke <init> method");
                }
                AccountState accountState =
repository.getAccountState(programInvoke.getContractAddress());
                if (accountState == null) {
                    return revert(String.format("contract[%s] does not exist",
programInvoke.getAddress()));
                }
                logTime("load account state");
                if (accountState.getNonce().compareTo(BigInteger.ZERO) <= 0) {
                    return revert(String.format("contract[%s] has stopped", programInvoke.getAddress()));
                }
```

```
            byte[] codes = repository.getCode(programInvoke.getContractAddress());
            classCodes = ClassCodeLoader.loadJarCache(codes);
            logTime("load code");
        }


        VM vm = VMFactory.createVM();
        logTime("load vm");

        vm.heap.loadClassCodes(classCodes);
        vm.methodArea.loadClassCodes(classCodes);

        logTime("load classes");

        ClassCode contractClassCode = getContractClassCode(classCodes);
        String methodDesc = ProgramDescriptors.parseDesc(programInvoke.getMethodDesc());
        MethodCode methodCode = vm.methodArea.loadMethod(contractClassCode.name,
programInvoke.getMethodName(), methodDesc);

        if (methodCode == null) {
            return revert(String.format("can't find method %s%s", programInvoke.getMethodName(),
programInvoke.getMethodDesc() == null ? "" : programInvoke.getMethodDesc()));
        }
        if (!methodCode.isPublic) {
            return revert("can only invoke public method");
        }
        if (!methodCode.hasPayableAnnotation() &&
programInvoke.getValue().compareTo(BigInteger.ZERO) > 0) {
            return revert("not a payable method");
        }
        if (methodCode.argsVariableType.size() != programInvoke.getArgs().length) {
            return revert(String.format("require %s parameters in method [%s%s]",
                    methodCode.argsVariableType.size(), methodCode.name,
methodCode.normalDesc));
        }

        logTime("load method");

        ObjectRef objectRef;
        if (programInvoke.isCreate()) {
            objectRef = vm.heap.newContract(programInvoke.getContractAddress(),
contractClassCode, repository);
```

```
        } else {
            objectRef = vm.heap.loadContract(programInvoke.getContractAddress(),
contractClassCode, repository);
        }

        logTime("load contract ref");

        if (programInvoke.getValue().compareTo(BigInteger.ZERO) > 0) {
getAccount(programInvoke.getContractAddress()).addBalance(programInvoke.getValue());
}
        vm.setProgramExecutor(this);
        vm.setRepository(repository);
        vm.setGas(programInvoke.getGasLimit());
        vm.addGasUsed(programInvoke.getData() == null ? 0 : programInvoke.getData().length);

        logTime("load end");

        vm.run(objectRef, methodCode, vmContext, programInvoke);

        logTime("run");

        ProgramResult programResult = new ProgramResult();
        programResult.setGasUsed(vm.getGasUsed());

        Result vmResult = vm.getResult();
        Object resultValue = vmResult.getValue();
        if (vmResult.isError() || vmResult.isException()) {
            if (resultValue != null && resultValue instanceof ObjectRef) {
                vm.setResult(new Result());
                String error = vm.heap.runToString((ObjectRef) resultValue);
                String stackTrace = vm.heap.stackTrace((ObjectRef) resultValue);
                programResult.error(error);
                programResult.setStackTrace(stackTrace);
            } else {
                programResult.error(null);
            }

            logTime("contract exception");

            this.revert = true;

            programResult.setGasUsed(vm.getGasUsed());
```

```
            return programResult;
        }

        repository.increaseNonce(programInvoke.getContractAddress());
        programResult.setNonce(repository.getNonce(programInvoke.getContractAddress()));
        programResult.setTransfers(vm.getTransfers());
        programResult.setEvents(vm.getEvents());
programResult.setBalance(getAccount(programInvoke.getContractAddress()).getBalance());
 if (resultValue != null) {
            if (resultValue instanceof ObjectRef) {
                String result = vm.heap.runToString((ObjectRef) resultValue);
                programResult.setResult(result);
            } else {
                programResult.setResult(resultValue.toString());
            }
        }

        if (methodCode.isPublic && methodCode.hasViewAnnotation()) {
            this.revert = true;
            programResult.view();
            programResult.setGasUsed(vm.getGasUsed());
            return programResult;
        }

        logTime("contract return");

        Map<DataWord, DataWord> contractState = vm.heap.contractState();
        logTime("contract state");

        for (Map.Entry<DataWord, DataWord> entry : contractState.entrySet()) {
            DataWord key = entry.getKey();
            DataWord value = entry.getValue();
            repository.addStorageRow(programInvoke.getContractAddress(), key, value);
        }
        logTime("add contract state");

        programResult.setGasUsed(vm.getGasUsed());

        return programResult;
    } catch (ErrorException e) {
        this.revert = true;
```

```
            //log.error("", e);
            ProgramResult programResult = new ProgramResult();
            programResult.setGasUsed(e.getGasUsed());
            //programResult.setStackTrace(e.getStackTraceMessage());
            logTime("error");
            return programResult.error(e.getMessage());
        } catch (RevertException e) {
            //log.error("", e);
            return revert(e.getMessage());
            //return revert(e.getMessage(), e.getStackTraceMessage());
        } catch (Exception e) {
            log.error("", e);
            ProgramResult programResult = revert(e.getMessage());
            //programResult.setStackTrace(ExceptionUtils.getStackTrace(e));
            return programResult;
        }
    }

    private ProgramResult revert(String errorMessage) {
        return revert(errorMessage, null);
    }

    private ProgramResult revert(String errorMessage, String stackTrace) {
        this.revert = true;
        ProgramResult programResult = new ProgramResult();
        programResult.setStackTrace(stackTrace);
        logTime("revert");
        return programResult.revert(errorMessage);
    }

    @Override
    public ProgramResult stop(byte[] address, byte[] sender) {
        checkThread();
        AccountState accountState = repository.getAccountState(address);
        if (accountState == null) {
            return revert("can't find contract");
        }
        if (!FastByteComparisons.equal(sender, accountState.getOwner())) {
            return revert("only the owner can stop the contract");
        }
        BigInteger balance = getTotalBalance(address, null);
        if (BigInteger.ZERO.compareTo(balance) != 0) {
```

```
            return revert("contract balance is not zero");
        }
        if (BigInteger.ZERO.compareTo(accountState.getNonce()) >= 0) {
            return revert("contract has stopped");
        }

        repository.setNonce(address, BigInteger.ZERO);

        ProgramResult programResult = new ProgramResult();

        return programResult;
    }

    @Override
    public ProgramStatus status(byte[] address) {
        checkThread();
        this.revert = true;
        AccountState accountState = repository.getAccountState(address);
        if (accountState == null) {
            return ProgramStatus.not_found;
        } else {
            BigInteger nonce = repository.getNonce(address);
            if (BigInteger.ZERO.compareTo(nonce) >= 0) {
                return ProgramStatus.stop;
            } else {
                return ProgramStatus.normal;
            }
        }
    }

    @Override
    public ProgramAccount getAccount(byte[] address) {
        ByteArrayWrapper addressWrapper = new ByteArrayWrapper(address);
        ProgramAccount account = accounts.get(addressWrapper);
        if (account == null) {
            BigInteger balance = getBalance(address, blockNumber);
            account = new ProgramAccount(address, balance);
            accounts.put(addressWrapper, account);
        }
        return account;
    }
```

```
    private BigInteger getBalance(byte[] address, Long blockNumber) {
        BigInteger balance = BigInteger.ZERO;
        if (vmContext != null) {
            balance = vmContext.getBalance(address, blockNumber);
        }
        return balance;
    }

    private BigInteger getTotalBalance(byte[] address, Long blockNumber) {
        BigInteger balance = BigInteger.ZERO;
        if (vmContext != null) {
            balance = vmContext.getTotalBalance(address, blockNumber);
        }
        return balance;
    }

    @Override
    public List<ProgramMethod> method(byte[] address) {
        checkThread();
        this.revert = true;
        byte[] codes = repository.getCode(address);
        return jarMethod(codes);
    }

    @Override
    public List<ProgramMethod> jarMethod(byte[] jarData) {
        this.revert = true;
        if (jarData == null || jarData.length < 1) {
            return new ArrayList<>();
        }
        Map<String, ClassCode> classCodes = ClassCodeLoader.loadJarCache(jarData);
        return getProgramMethods(classCodes);
    }

    private void checkThread() {
        if (thread == null) {
            throw new RuntimeException("must use the begin method");
        }
        Thread currentThread = Thread.currentThread();
        if (!currentThread.equals(thread)) {
            throw new RuntimeException(String.format("method must be executed in %s, current %s",
thread, currentThread));
```

```
        }
    }

    private static List<ProgramMethod> getProgramMethods(Map<String, ClassCode> classCodes)
{
        List<ProgramMethod> programMethods =
getProgramMethodCodes(classCodes).stream().map(methodCode -> {
            ProgramMethod method = new ProgramMethod();
            method.setName(methodCode.name);
            method.setDesc(methodCode.normalDesc);
            method.setArgs(methodCode.args);
            method.setReturnArg(methodCode.returnArg);
            method.setView(methodCode.hasViewAnnotation());
            method.setPayable(methodCode.hasPayableAnnotation());
            method.setEvent(false);
            return method;
        }).collect(Collectors.toList());
        programMethods.addAll(getEventConstructor(classCodes));
        return programMethods;
    }

    public static List<MethodCode> getProgramMethodCodes(Map<String, ClassCode>
classCodes) {
        Map<String, MethodCode> methodCodes = new LinkedHashMap<>();
        ClassCode contractClassCode = getContractClassCode(classCodes);
        if (contractClassCode != null) {
            contractMethods(methodCodes, classCodes, contractClassCode, false);
        }
        return methodCodes.values().stream().collect(Collectors.toList());
    }

    private static ClassCode getContractClassCode(Map<String, ClassCode> classCodes) {
        return classCodes.values().stream().filter(classCode ->
classCode.interfaces.contains(ProgramConstants.CONTRACT_INTERFACE_NAME)).findFirst().o
rElse(null);
    }

    private static void contractMethods(Map<String, MethodCode> methodCodes, Map<String,
ClassCode> classCodes, ClassCode classCode, boolean isSupperClass) {
        classCode.methods.stream().filter(methodCode -> {
            if (methodCode.isPublic && !methodCode.isAbstract) {
                return true;
```

```
            } else {
                return false;
            }
        }).forEach(methodCode -> {
            if (isSupperClass && Constants.CONSTRUCTOR_NAME.equals(methodCode.name)) {
            } else if (Constants.CLINIT_NAME.equals(methodCode.name)) {
            } else {
                String name = methodCode.name + "." + methodCode.desc;
                methodCodes.putIfAbsent(name, methodCode);
            }
        });
        String superName = classCode.superName;
        if (StringUtils.isNotEmpty(superName)) {
            classCodes.values().stream().filter(code -> superName.equals(code.name)).findFirst()
                    .ifPresent(code -> {
                        contractMethods(methodCodes, classCodes, code, true);
                    });
        }
    }

    private static Set<ProgramMethod> getEventConstructor(Map<String, ClassCode> classCodes)
{
        Map<String, MethodCode> methodCodes = new LinkedHashMap<>();
        getEventClassCodes(classCodes).forEach(classCode -> {
            for (MethodCode methodCode : classCode.methods) {
                if (methodCode.isConstructor) {
                    methodCodes.put(methodCode.fullName, methodCode);
                }
            }
        });
        return methodCodes.values().stream()
                .filter(methodCode -> methodCode.isConstructor)
                .map(methodCode -> {
                    ProgramMethod method = new ProgramMethod();
                    method.setName(methodCode.classCode.simpleName);
                    method.setDesc(methodCode.normalDesc);
                    method.setArgs(methodCode.args);
                    method.setReturnArg(methodCode.returnArg);
                    method.setView(methodCode.hasViewAnnotation());
                    method.setPayable(methodCode.hasPayableAnnotation());
                    method.setEvent(true);
                    return method;
```

```
            }).collect(Collectors.toSet());
    }

    private static List<ClassCode> getEventClassCodes(Map<String, ClassCode> classCodes) {
        ClassCodes allCodes = new ClassCodes(classCodes);
        return classCodes.values().stream().filter(classCode -> !classCode.isAbstract
                && allCodes.instanceOf(classCode, ProgramConstants.EVENT_INTERFACE_NAME))
                .collect(Collectors.toList());
    }

    private static Source<byte[], byte[]> stateSource(DBService dbService) {
        LevelDbDataSource.dbService = dbService;
        SystemProperties config = SystemProperties.getDefault();
        CommonConfig commonConfig = CommonConfig.getDefault();
        StateSource stateSource = commonConfig.stateSource();
        stateSource.setConfig(config);
        stateSource.setCommonConfig(commonConfig);
        return stateSource;
    }

    public void logTime(String message) {
        if (log.isDebugEnabled()) {
            long currentTime = System.currentTimeMillis();
            long step = currentTime - this.currentTime;
            long runtime = currentTime - this.beginTime;
            this.currentTime = currentTime;
            ProgramTime.cache.putIfAbsent(message, new ProgramTime());
            ProgramTime time = ProgramTime.cache.get(message);
            time.add(step);
            log.debug("[{}] runtime: {}ms, step: {}ms, {}", message, runtime, step, time);
        }
//        if (step > 100) {
//            List<String> list = new ArrayList<>();
//            list.add(String.format("%s, runtime: %sms, step: %sms", message, runtime, step));
//            try {
//                FileUtils.writeLines(new File("/tmp/long.log"), list, true);
//            } catch (IOException e) {
//                log.error("", e);
//            }
//        }
    }
```

}

99:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramInvoke.java
 */
package io.nuls.contract.vm.program.impl;

import java.math.BigInteger;

public class ProgramInvoke {

    /**
     *
     *
     */
    private byte[] contractAddress;

    private String address;

    /**
     *
     */
    private byte[] sender;

    /**
     * gas
     */
    private long price;

    /**
     * gas
     */
    private long gasLimit;

    /**
     *
     */
    private BigInteger value;

    /**
     *
     */

```
private long number;

/**
 *
 */
private byte[] data;

/**
 *
 */
private String methodName;

/**
 *
 */
private String methodDesc;

/**
 *
 */
private String[][] args;

/**
 * Gas
 */
private boolean estimateGas;

private boolean create;

private boolean internalCall;

public byte[] getContractAddress() {
    return contractAddress;
}

public void setContractAddress(byte[] contractAddress) {
    this.contractAddress = contractAddress;
}

public String getAddress() {
    return address;
}
```

```
public void setAddress(String address) {
    this.address = address;
}

public byte[] getSender() {
    return sender;
}

public void setSender(byte[] sender) {
    this.sender = sender;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public long getGasLimit() {
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public BigInteger getValue() {
    return value;
}

public void setValue(BigInteger value) {
    this.value = value;
}

public long getNumber() {
    return number;
}

public void setNumber(long number) {
```

```
    this.number = number;
}

public byte[] getData() {
    return data;
}

public void setData(byte[] data) {
    this.data = data;
}

public String getMethodName() {
    return methodName;
}

public void setMethodName(String methodName) {
    this.methodName = methodName;
}

public String getMethodDesc() {
    return methodDesc;
}

public void setMethodDesc(String methodDesc) {
    this.methodDesc = methodDesc;
}

public String[][] getArgs() {
    return args;
}

public void setArgs(String[][] args) {
    this.args = args;
}

public boolean isEstimateGas() {
    return estimateGas;
}

public void setEstimateGas(boolean estimateGas) {
    this.estimateGas = estimateGas;
}
```

```
    public boolean isCreate() {
        return create;
    }

    public void setCreate(boolean create) {
        this.create = create;
    }

    public boolean isInternalCall() {
        return internalCall;
    }

    public void setInternalCall(boolean internalCall) {
        this.internalCall = internalCall;
    }

}

100:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\impl\ProgramTime.java
 */
package io.nuls.contract.vm.program.impl;

import java.util.HashMap;
import java.util.Map;

public class ProgramTime {

    public static final Map<String, ProgramTime> cache = new HashMap<>(1024);

    private long num;
    private long total;
    private long average;

    public void add(long time) {
        this.num += 1;
        this.total += time;
        this.average = this.total / this.num;
    }

    public long getNum() {
```

```
        return num;
    }

    public void setNum(long num) {
        this.num = num;
    }

    public long getTotal() {
        return total;
    }

    public void setTotal(long total) {
        this.total = total;
    }

    public long getAverage() {
        return average;
    }

    public void setAverage(long average) {
        this.average = average;
    }

    @Override
    public String toString() {
        return "ProgramTime{" +
                "num=" + num +
                ", total=" + total +
                ", average=" + average +
                '}';
    }

}
```

101:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\program\ProgramAccount.java

```
    public BigInteger addBalance(BigInteger value) {
        balance = balance.add(value);
        return balance;
    }

}
```

```
102:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramCall.java
 */
package io.nuls.contract.vm.program;

import io.nuls.kernel.utils.AddressTool;

import java.math.BigInteger;
import java.util.Arrays;

import static io.nuls.contract.util.ContractUtil.argToString;

public class ProgramCall {

    /**
     *
     */
    private long number;

    /**
     *
     */
    private byte[] sender;

    /**
     *
     */
    private BigInteger value;

    /**
     * Gas
     */
    private long gasLimit;

    /**
     *
     */
    private long price;

    /**
     *
```

```
 */
private byte[] contractAddress;

/**
 *
 */
private String methodName;

/**
 *
 */
private String methodDesc;

/**
 *
 */
private String[][] args;

/**
 * Gas
 */
private boolean estimateGas;

private boolean internalCall;

public void args(String... args) {
    setArgs(args);
}

public String[][] getArgs() {
    return args;
}

public void setArgs(String[][] args) {
    this.args = args;
}

public void setArgs(String[] args) {
    this.args = twoDimensionalArray(args);
}

public static String[][] twoDimensionalArray(String[] args) {
```

```
    if (args == null) {
        return null;
    } else {
        String[][] two = new String[args.length][0];
        for (int i = 0; i < args.length; i++) {
            String arg = args[i];
            if (arg != null) {
                two[i] = new String[]{arg};
            }
        }
        return two;
    }
}

public ProgramCall() {
}

public long getNumber() {
    return number;
}

public void setNumber(long number) {
    this.number = number;
}

public byte[] getSender() {
    return sender;
}

public void setSender(byte[] sender) {
    this.sender = sender;
}

public BigInteger getValue() {
    return value;
}

public void setValue(BigInteger value) {
    this.value = value;
}

public long getGasLimit() {
```

```
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public byte[] getContractAddress() {
    return contractAddress;
}

public void setContractAddress(byte[] contractAddress) {
    this.contractAddress = contractAddress;
}

public String getMethodName() {
    return methodName;
}

public void setMethodName(String methodName) {
    this.methodName = methodName;
}

public String getMethodDesc() {
    return methodDesc;
}

public void setMethodDesc(String methodDesc) {
    this.methodDesc = methodDesc;
}

public boolean isEstimateGas() {
    return estimateGas;
}
```

```
public void setEstimateGas(boolean estimateGas) {
    this.estimateGas = estimateGas;
}

public boolean isInternalCall() {
    return internalCall;
}

public void setInternalCall(boolean internalCall) {
    this.internalCall = internalCall;
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    ProgramCall that = (ProgramCall) o;

    if (number != that.number) {
        return false;
    }
    if (gasLimit != that.gasLimit) {
        return false;
    }
    if (price != that.price) {
        return false;
    }
    if (estimateGas != that.estimateGas) {
        return false;
    }
    if (!Arrays.equals(sender, that.sender)) {
        return false;
    }
    if (value != null ? !value.equals(that.value) : that.value != null) {
        return false;
    }
```

```
        if (!Arrays.equals(contractAddress, that.contractAddress)) {
            return false;
        }
        if (methodName != null ? !methodName.equals(that.methodName) : that.methodName !=
null) {
            return false;
        }
        if (methodDesc != null ? !methodDesc.equals(that.methodDesc) : that.methodDesc != null) {
            return false;
        }
        // Probably incorrect - comparing Object[] arrays with Arrays.equals
        return Arrays.equals(args, that.args);
    }

    @Override
    public int hashCode() {
        int result = (int) (number ^ (number >>> 32));
        result = 31 * result + Arrays.hashCode(sender);
        result = 31 * result + (value != null ? value.hashCode() : 0);
        result = 31 * result + (int) (gasLimit ^ (gasLimit >>> 32));
        result = 31 * result + (int) (price ^ (price >>> 32));
        result = 31 * result + Arrays.hashCode(contractAddress);
        result = 31 * result + (methodName != null ? methodName.hashCode() : 0);
        result = 31 * result + (methodDesc != null ? methodDesc.hashCode() : 0);
        result = 31 * result + Arrays.hashCode(args);
        result = 31 * result + (estimateGas ? 1 : 0);
        return result;
    }

    @Override
    public String toString() {
        return "ProgramCall{" +
                "number=" + number +
                ", sender=" + (sender != null ? AddressTool.getStringAddressByBytes(sender) : sender)
+
                ", value=" + value +
                ", gasLimit=" + gasLimit +
                ", price=" + price +
                ", contractAddress=" + (contractAddress != null ?
AddressTool.getStringAddressByBytes(contractAddress) : contractAddress) +
                ", methodName=" + methodName +
                ", methodDesc=" + methodDesc +
```

```
                ", args=" + argToString(args) +
                ", estimateGas=" + estimateGas +
                '}';
    }
}

103:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramCreate.java
 */
package io.nuls.contract.vm.program;

import io.nuls.kernel.utils.AddressTool;

import java.math.BigInteger;
import java.util.Arrays;

import static io.nuls.contract.util.ContractUtil.argToString;

public class ProgramCreate {

    /**
     *
     */
    private long number;

    /**
     *
     */
    private byte[] sender;

    /**
     *
     */
    private BigInteger value;

    /**
     * Gas
     */
    private long gasLimit;

    /**
     *
```

```
 */
private long price;

/**
 *
 */
private byte[] contractAddress;

/**
 *
 */
private byte[] contractCode;

/**
 *
 */
private String[][] args;

/**
 * Gas
 */
private boolean estimateGas;

public void args(String... args) {
    setArgs(args);
}

public String[][] getArgs() {
    return args;
}

public void setArgs(String[][] args) {
    this.args = args;
}

public void setArgs(String[] args) {
    this.args = ProgramCall.twoDimensionalArray(args);
}

public ProgramCreate() {
}
```

```
public long getNumber() {
    return number;
}

public void setNumber(long number) {
    this.number = number;
}

public byte[] getSender() {
    return sender;
}

public void setSender(byte[] sender) {
    this.sender = sender;
}

public BigInteger getValue() {
    return value;
}

public void setValue(BigInteger value) {
    this.value = value;
}

public long getGasLimit() {
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public byte[] getContractAddress() {
    return contractAddress;
```

```
}

public void setContractAddress(byte[] contractAddress) {
    this.contractAddress = contractAddress;
}

public byte[] getContractCode() {
    return contractCode;
}

public void setContractCode(byte[] contractCode) {
    this.contractCode = contractCode;
}

public boolean isEstimateGas() {
    return estimateGas;
}

public void setEstimateGas(boolean estimateGas) {
    this.estimateGas = estimateGas;
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    ProgramCreate that = (ProgramCreate) o;

    if (number != that.number) {
        return false;
    }
    if (gasLimit != that.gasLimit) {
        return false;
    }
    if (price != that.price) {
        return false;
    }
```

```
        if (estimateGas != that.estimateGas) {
            return false;
        }
        if (!Arrays.equals(sender, that.sender)) {
            return false;
        }
        if (value != null ? !value.equals(that.value) : that.value != null) {
            return false;
        }
        if (!Arrays.equals(contractAddress, that.contractAddress)) {
            return false;
        }
        if (!Arrays.equals(contractCode, that.contractCode)) {
            return false;
        }
        // Probably incorrect - comparing Object[] arrays with Arrays.equals
        return Arrays.equals(args, that.args);
    }

    @Override
    public int hashCode() {
        int result = (int) (number ^ (number >>> 32));
        result = 31 * result + Arrays.hashCode(sender);
        result = 31 * result + (value != null ? value.hashCode() : 0);
        result = 31 * result + (int) (gasLimit ^ (gasLimit >>> 32));
        result = 31 * result + (int) (price ^ (price >>> 32));
        result = 31 * result + Arrays.hashCode(contractAddress);
        result = 31 * result + Arrays.hashCode(contractCode);
        result = 31 * result + Arrays.hashCode(args);
        result = 31 * result + (estimateGas ? 1 : 0);
        return result;
    }

    @Override
    public String toString() {
        return "ProgramCreate{" +
                "number=" + number +
                ", sender=" + (sender != null ? AddressTool.getStringAddressByBytes(sender) : sender)
+
                ", value=" + value +
                ", gasLimit=" + gasLimit +
                ", price=" + price +
```

```
            ", contractAddress=" + (contractAddress != null ?
AddressTool.getStringAddressByBytes(contractAddress) : contractAddress) +
            ", contractCode=" + (contractCode != null ? String.valueOf(contractCode.length) : 0) +
            ", args=" + argToString(args) +
            ", estimateGas=" + estimateGas +
            '}';
    }
}

104:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramExecutor.java
 */
package io.nuls.contract.vm.program;

import java.util.List;

public interface ProgramExecutor {

    ProgramExecutor begin(byte[] prevStateRoot);

    ProgramExecutor startTracking();

    void commit();

    byte[] getRoot();

    ProgramResult create(ProgramCreate programCreate);

    ProgramResult call(ProgramCall programCall);

    ProgramResult stop(byte[] address, byte[] sender);

    List<ProgramMethod> method(byte[] address);

    List<ProgramMethod> jarMethod(byte[] jarData);

    ProgramStatus status(byte[] address);

    ProgramAccount getAccount(byte[] address);

}
```

```
105:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramMethod.java
 */
package io.nuls.contract.vm.program;

import java.util.List;

public class ProgramMethod {

    private String name;

    private String desc;

    private List<ProgramMethodArg> args;

    private String returnArg;

    private boolean view;

    private boolean event;

    private boolean payable;

    public ProgramMethod() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }
```

```
public List<ProgramMethodArg> getArgs() {
    return args;
}

public void setArgs(List<ProgramMethodArg> args) {
    this.args = args;
}

public String getReturnArg() {
    return returnArg;
}

public void setReturnArg(String returnArg) {
    this.returnArg = returnArg;
}

public boolean isView() {
    return view;
}

public void setView(boolean view) {
    this.view = view;
}

public boolean isEvent() {
    return event;
}

public void setEvent(boolean event) {
    this.event = event;
}

public boolean isPayable() {
    return payable;
}

public void setPayable(boolean payable) {
    this.payable = payable;
}

@Override
public boolean equals(Object o) {
```

```
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    ProgramMethod that = (ProgramMethod) o;

    if (view != that.view) {
        return false;
    }
    if (event != that.event) {
        return false;
    }
    if (payable != that.payable) {
        return false;
    }
    if (name != null ? !name.equals(that.name) : that.name != null) {
        return false;
    }
    if (desc != null ? !desc.equals(that.desc) : that.desc != null) {
        return false;
    }
    if (args != null ? !args.equals(that.args) : that.args != null) {
        return false;
    }
    return returnArg != null ? returnArg.equals(that.returnArg) : that.returnArg == null;
}

public boolean equalsNrc20Method(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    ProgramMethod that = (ProgramMethod) o;

    if (view != that.view) {
        return false;
```

```
    }
    if (event != that.event) {
        return false;
    }
    if (name != null ? !name.equals(that.name) : that.name != null) {
        return false;
    }
    if (args != null) {
        if(that.args == null) {
            return false;
        }
        if(!isEqualNrc20Args(args, that.args)) {
            return false;
        }
    } else {
        if(that.args != null) {
            return false;
        }
    }
    return returnArg != null ? returnArg.equals(that.returnArg) : that.returnArg == null;
}

public String[] argsType2Array() {
    if(args != null && args.size() > 0) {
        int size = args.size();
        String[] result = new String[size];
        for(int i = 0; i < size; i++) {
            result[i] = args.get(i).getType();
        }
        return result;
    } else {
        return null;
    }
}

private boolean isEqualNrc20Args(List<ProgramMethodArg> a, List<ProgramMethodArg> b) {
    if (a.size() != b.size()) {
        return false;
    } else {
        /*
        //
        Map<String, ProgramMethodArg> mapA =
```

```
a.stream().collect(Collectors.toMap(ProgramMethodArg::getName, Function.identity(), (key1,
key2) -> key2, LinkedHashMap::new));
        Map<String, ProgramMethodArg> mapB =
b.stream().collect(Collectors.toMap(ProgramMethodArg::getName, Function.identity(), (key1,
key2) -> key2, LinkedHashMap::new));
        Set<Map.Entry<String, ProgramMethodArg>> entriesA = mapA.entrySet();
        String methodName;
        ProgramMethodArg methodArg;
        for(Map.Entry<String, ProgramMethodArg> entryA : entriesA) {
            methodName = entryA.getKey();
            if(!mapB.containsKey(methodName)) {
                return false;
            }
            methodArg = entryA.getValue();
            if(!methodArg.equalsNrc20(mapB.get(methodName))) {
                return false;
            }
        }
        */

        //
        int size = a.size();
        ProgramMethodArg argA, argB;
        for(int i = 0; i < size; i++){
            argA = a.get(i);
            argB = b.get(i);
            if(!argA.equalsNrc20(argB)) {
                return false;
            }
        }
        return true;
    }
}

@Override
public int hashCode() {
    int result = name != null ? name.hashCode() : 0;
    result = 31 * result + (desc != null ? desc.hashCode() : 0);
    result = 31 * result + (args != null ? args.hashCode() : 0);
    result = 31 * result + (returnArg != null ? returnArg.hashCode() : 0);
    result = 31 * result + (view ? 1 : 0);
    result = 31 * result + (event ? 1 : 0);
```

```
        result = 31 * result + (payable ? 1 : 0);
        return result;
    }

    @Override
    public String toString() {
        return "ProgramMethod{" +
                "name=" + name +
                ", desc=" + desc +
                ", args=" + args +
                ", returnArg=" + returnArg +
                ", view=" + view +
                ", event=" + event +
                ", payable=" + payable +
                '}';
    }

}

106:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramMethodArg.java
 */
package io.nuls.contract.vm.program;

public class ProgramMethodArg {

    private String type;

    private String name;

    private boolean required;

    public ProgramMethodArg() {
    }

    public ProgramMethodArg(String type, String name, boolean required) {
        this.type = type;
        this.name = name;
        this.required = required;
    }

    public String getType() {
```

```
    return type;
}

public void setType(String type) {
    this.type = type;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public boolean isRequired() {
    return required;
}

public void setRequired(boolean required) {
    this.required = required;
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    ProgramMethodArg that = (ProgramMethodArg) o;

    if (required != that.required) {
        return false;
    }
    if (type != null ? !type.equals(that.type) : that.type != null) {
        return false;
    }
    return name != null ? name.equals(that.name) : that.name == null;
}
```

```
    public boolean equalsNrc20(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        ProgramMethodArg that = (ProgramMethodArg) o;

        if (required != that.required) {
            return false;
        }
        return type != null ? type.equals(that.type) : that.type == null;
    }

    @Override
    public int hashCode() {
        int result = type != null ? type.hashCode() : 0;
        result = 31 * result + (name != null ? name.hashCode() : 0);
        result = 31 * result + (required ? 1 : 0);
        return result;
    }

    @Override
    public String toString() {
        return "ProgramMethodArg{" +
                "type=" + type +
                ", name=" + name +
                ", required=" + required +
                '}';
    }

}

107:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramResult.java
 */
package io.nuls.contract.vm.program;

import java.math.BigInteger;
```

```
import java.util.ArrayList;
import java.util.List;

public class ProgramResult {

    private long gasUsed;

    private String result;

    private boolean revert;

    private boolean error;

    private String errorMessage;

    private String stackTrace;

    private BigInteger balance;

    private BigInteger nonce;

    private List<ProgramTransfer> transfers = new ArrayList<>();

    private List<String> events = new ArrayList<>();

    public ProgramResult revert(String errorMessage) {
        this.revert = true;
        this.errorMessage = errorMessage;
        return this;
    }

    public ProgramResult error(String errorMessage) {
        this.error = true;
        this.errorMessage = errorMessage;
        return this;
    }

    public void view() {
        this.transfers = new ArrayList<>();
        this.events = new ArrayList<>();
    }
```

```
public ProgramResult() {
}

public boolean isSuccess() {
    return !error && !revert;
}

public long getGasUsed() {
    return gasUsed;
}

public void setGasUsed(long gasUsed) {
    this.gasUsed = gasUsed;
}

public String getResult() {
    return result;
}

public void setResult(String result) {
    this.result = result;
}

public boolean isRevert() {
    return revert;
}

public void setRevert(boolean revert) {
    this.revert = revert;
}

public boolean isError() {
    return error;
}

public void setError(boolean error) {
    this.error = error;
}

public String getErrorMessage() {
    return errorMessage;
}
```

```
public void setErrorMessage(String errorMessage) {
    this.errorMessage = errorMessage;
}

public String getStackTrace() {
    return stackTrace;
}

public void setStackTrace(String stackTrace) {
    this.stackTrace = stackTrace;
}

public BigInteger getBalance() {
    return balance;
}

public void setBalance(BigInteger balance) {
    this.balance = balance;
}

public BigInteger getNonce() {
    return nonce;
}

public void setNonce(BigInteger nonce) {
    this.nonce = nonce;
}

public List<ProgramTransfer> getTransfers() {
    return transfers;
}

public void setTransfers(List<ProgramTransfer> transfers) {
    this.transfers = transfers;
}

public List<String> getEvents() {
    return events;
}

public void setEvents(List<String> events) {
```

```
        this.events = events;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        ProgramResult that = (ProgramResult) o;

        if (gasUsed != that.gasUsed) {
            return false;
        }
        if (revert != that.revert) {
            return false;
        }
        if (error != that.error) {
            return false;
        }
        if (result != null ? !result.equals(that.result) : that.result != null) {
            return false;
        }
        if (errorMessage != null ? !errorMessage.equals(that.errorMessage) : that.errorMessage !=
null) {
            return false;
        }
        if (stackTrace != null ? !stackTrace.equals(that.stackTrace) : that.stackTrace != null) {
            return false;
        }
        if (balance != null ? !balance.equals(that.balance) : that.balance != null) {
            return false;
        }
        if (nonce != null ? !nonce.equals(that.nonce) : that.nonce != null) {
            return false;
        }
        if (transfers != null ? !transfers.equals(that.transfers) : that.transfers != null) {
            return false;
        }
```

```
        return events != null ? events.equals(that.events) : that.events == null;
    }

    @Override
    public int hashCode() {
        int result1 = (int) (gasUsed ^ (gasUsed >>> 32));
        result1 = 31 * result1 + (result != null ? result.hashCode() : 0);
        result1 = 31 * result1 + (revert ? 1 : 0);
        result1 = 31 * result1 + (error ? 1 : 0);
        result1 = 31 * result1 + (errorMessage != null ? errorMessage.hashCode() : 0);
        result1 = 31 * result1 + (stackTrace != null ? stackTrace.hashCode() : 0);
        result1 = 31 * result1 + (balance != null ? balance.hashCode() : 0);
        result1 = 31 * result1 + (nonce != null ? nonce.hashCode() : 0);
        result1 = 31 * result1 + (transfers != null ? transfers.hashCode() : 0);
        result1 = 31 * result1 + (events != null ? events.hashCode() : 0);
        return result1;
    }

    @Override
    public String toString() {
        return "ProgramResult{" +
                "gasUsed=" + gasUsed +
                ", result=" + result +
                ", revert=" + revert +
                ", error=" + error +
                ", errorMessage=" + errorMessage +
                ", stackTrace=" + stackTrace +
                ", balance=" + balance +
                ", nonce=" + nonce +
                ", transfers=" + transfers +
                ", events=" + events +
                '}';
    }

}

108:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramStatus.java
 */
package io.nuls.contract.vm.program;

public enum ProgramStatus {
```

```
    not_found,

    normal,

    stop,

}

109:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\program\ProgramTransfer.java
 */
package io.nuls.contract.vm.program;

import java.math.BigInteger;
import java.util.Arrays;

public class ProgramTransfer {

    private byte[] from;

    private byte[] to;

    private BigInteger value;

    public ProgramTransfer(byte[] from, byte[] to, BigInteger value) {
        this.from = from;
        this.to = to;
        this.value = value;
    }

    public byte[] getFrom() {
        return from;
    }

    public byte[] getTo() {
        return to;
    }

    public BigInteger getValue() {
        return value;
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        ProgramTransfer that = (ProgramTransfer) o;

        if (!Arrays.equals(from, that.from)) {
            return false;
        }
        if (!Arrays.equals(to, that.to)) {
            return false;
        }
        return value != null ? value.equals(that.value) : that.value == null;
    }

    @Override
    public int hashCode() {
        int result = Arrays.hashCode(from);
        result = 31 * result + Arrays.hashCode(to);
        result = 31 * result + (value != null ? value.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "ProgramTransfer{" +
                "from=" + Arrays.toString(from) +
                ", to=" + Arrays.toString(to) +
                ", value=" + value +
                '}';
    }

}
```

110:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\Result.java

```
 */
package io.nuls.contract.vm;

import io.nuls.contract.vm.code.VariableType;

public class Result {

    private VariableType variableType;

    private Object value;

    private boolean ended;

    private boolean exception;

    private boolean error;

    public Result() {
    }

    public Result(VariableType variableType) {
        this.variableType = variableType;
    }

    public void value(Object value) {
        this.value = value;
        this.ended = true;
    }

    public void exception(ObjectRef exception) {
        //java.lang.Exception
        this.value(exception);
        this.variableType = exception.getVariableType();
        this.exception = true;
    }

    public void error(ObjectRef error) {
        //java.lang.Error
        this.value(error);
        this.variableType = error.getVariableType();
        this.error = true;
    }
```

```
    public VariableType getVariableType() {
        return variableType;
    }

    public Object getValue() {
        return value;
    }

    public boolean isEnded() {
        return ended;
    }

    public boolean isException() {
        return exception;
    }

    public boolean isError() {
        return error;
    }

    @Override
    public String toString() {
        return "Result{" +
                "variableType=" + variableType +
                ", value=" + value +
                ", ended=" + ended +
                ", exception=" + exception +
                ", error=" + error +
                '}';
    }

}

111:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\util\CloneUtils.java
 */
package io.nuls.contract.vm.util;

import io.nuls.contract.vm.ObjectRef;

import java.lang.reflect.Array;
```

```
import java.util.LinkedHashMap;
import java.util.Map;

import static io.nuls.contract.vm.util.Utils.hashMapInitialCapacity;

public class CloneUtils {

    public static void clone(Map<String, Object> source, Map<String, Object> target) {
        for (Map.Entry<String, Object> entry : source.entrySet()) {
            String key = entry.getKey();
            Object object = entry.getValue();
            Object newObject = cloneObject(object);
            target.put(key, newObject);
        }
    }

    public static Map<String, Object> clone(Map<String, Object> source) {
        Map<String, Object> target = new LinkedHashMap<>(hashMapInitialCapacity(source.size()));
        clone(source, target);
        return target;
    }

    public static Object cloneObject(Object object) {
        Object newObject = null;
        if (object == null) {
            newObject = null;
        } else if (object instanceof Integer) {
            newObject = ((Integer) object).intValue();
        } else if (object instanceof Long) {
            newObject = ((Long) object).longValue();
        } else if (object instanceof Float) {
            newObject = ((Float) object).floatValue();
        } else if (object instanceof Double) {
            newObject = ((Double) object).doubleValue();
        } else if (object instanceof Boolean) {
            newObject = ((Boolean) object).booleanValue();
        } else if (object instanceof Byte) {
            newObject = ((Byte) object).byteValue();
        } else if (object instanceof Character) {
            newObject = ((Character) object).charValue();
        } else if (object instanceof Short) {
            newObject = ((Short) object).shortValue();
```

```
        } else if (object instanceof String) {
            newObject = object;
        } else if (object instanceof ObjectRef) {
            newObject = object;
        } else if (object.getClass().isArray()) {
            int length = Array.getLength(object);
            Object array = Array.newInstance(object.getClass().getComponentType(), length);
            System.arraycopy(object, 0, array, 0, length);
            newObject = array;
        } else {
            newObject = object;
        }
        return newObject;
    }

}

112:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\util\Constants.java
 */
package io.nuls.contract.vm.util;

public class Constants {

    public static final String DOLLAR = "$";

    public static final String CLASS_SEPARATOR = "/";

    public static final String CLASS_SUFFIX = ".class";

    public static final String CLINIT_NAME = "<clinit>";

    public static final String CLINIT_DESC = "()V";

    public static final String CONSTRUCTOR_NAME = "<init>";

    public static final String ARRAY_START = "[";

    public static final String ARRAY_PREFIX = "[L";

    public static final String ARRAY_SUFFIX = ";";
```

```
    public static final String OBJECT_CLASS_NAME = "java/lang/Object";

    public static final String OBJECT_CLASS_DESC = "Ljava/lang/Object;";

    public static final String TO_STRING_METHOD_NAME = "toString";

    public static final String TO_STRING_METHOD_DESC = "()Ljava/lang/String;";

    public static final String CLONE_METHOD_NAME = "clone";

    public static final String CLONE_METHOD_DESC = "()Ljava/lang/Object;";

    public static final String HASH = "hash";

    public static final String VALUE = "value";

    public static final String CHAR_CONSTRUCTOR_DESC = "(C)V";

    public static final String BYTES_CONSTRUCTOR_DESC = "([B)V";

    public static final String CONSTRUCTOR_DESC = "()V";

    public static final String CONSTRUCTOR_STRING_DESC = "(Ljava/lang/String;)V";

    public static final String CLASS_NAME = "java/lang/Class";

    public static final String CLASS_DESC = "Ljava/lang/Class;";

}
```

113:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\util\JsonUtils.java

```
 */
package io.nuls.contract.vm.util;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.type.ArrayType;
import com.fasterxml.jackson.databind.type.TypeFactory;
import com.google.common.collect.BiMap;
import io.nuls.contract.vm.ObjectRef;
```

```
import java.io.IOException;
import java.lang.reflect.Array;
import java.util.LinkedHashMap;
import java.util.Map;

import static io.nuls.contract.vm.util.Utils.hashMapInitialCapacity;

public class JsonUtils {

    private static final ObjectMapper OBJECT_MAPPER = new ObjectMapper();

    public static String toJson(Object value) {
        try {
            return OBJECT_MAPPER.writeValueAsString(value);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }

    public static <T> T toObject(String value, Class<T> valueType) {
        try {
            return OBJECT_MAPPER.readValue(value, valueType);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static <T> T toArray(String value, Class<?> elementType) {
        ArrayType arrayType = TypeFactory.defaultInstance().constructArrayType(elementType);
        try {
            return OBJECT_MAPPER.readValue(value, arrayType);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static String encodeArray(Object value, Class<?> elementType, BiMap<String, String>
classNames) {
        String json;
        if (elementType == ObjectRef.class) {
            int length = Array.getLength(value);
            String[] array = new String[length];
```

```
            for (int i = 0; i < length; i++) {
                ObjectRef objectRef = (ObjectRef) Array.get(value, i);
                if (objectRef != null) {
                    array[i] = objectRef.getEncoded(classNames);
                }
            }
            json = toJson(array);
        } else {
            json = toJson(value);
        }
        return json;
    }

    public static Object decodeArray(String value, Class<?> elementType, BiMap<String, String>
classNames) {
        if (elementType == ObjectRef.class) {
            Object array = toArray(value, String.class);
            int length = Array.getLength(array);
            ObjectRef[] objectRefs = new ObjectRef[length];
            for (int i = 0; i < length; i++) {
                String s = (String) Array.get(array, i);
                if (s != null) {
                    objectRefs[i] = new ObjectRef(s, classNames);
                }
            }
            return objectRefs;
        } else {
            return toArray(value, elementType);
        }
    }

    public static String encode(Object value, BiMap<String, String> classNames) {
        if (value == null) {
            return null;
        } else if (value.getClass().isArray()) {
            Class clazz = value.getClass().getComponentType();
            if (clazz == Integer.TYPE) {
                return "[I_" + encodeArray(value, clazz, classNames);
            } else if (clazz == Long.TYPE) {
                return "[J_" + encodeArray(value, clazz, classNames);
            } else if (clazz == Float.TYPE) {
                return "[F_" + encodeArray(value, clazz, classNames);
```

```
    } else if (clazz == Double.TYPE) {
        return "[D_" + encodeArray(value, clazz, classNames);
    } else if (clazz == Boolean.TYPE) {
        return "[Z_" + encodeArray(value, clazz, classNames);
    } else if (clazz == Byte.TYPE) {
        return "[B_" + encodeArray(value, clazz, classNames);
    } else if (clazz == Character.TYPE) {
        return "[C_" + encodeArray(value, clazz, classNames);
    } else if (clazz == Short.TYPE) {
        return "[S_" + encodeArray(value, clazz, classNames);
    } else {
        return "[R_" + encodeArray(value, clazz, classNames);
    }
} else if (value instanceof Map) {
    Map map = (Map) value;
    Map map1 = new LinkedHashMap(hashMapInitialCapacity(map.size()));
    map.forEach((k, v) -> {
        map1.put(k, encode(v, classNames));
    });
    return toJson(map1);
} else if (value instanceof Integer) {
    return "I_" + value;
} else if (value instanceof Long) {
    return "J_" + value;
} else if (value instanceof Float) {
    return "F_" + value;
} else if (value instanceof Double) {
    return "D_" + value;
} else if (value instanceof Boolean) {
    return "Z_" + value;
} else if (value instanceof Byte) {
    return "B_" + value;
} else if (value instanceof Character) {
    return "C_" + value;
} else if (value instanceof Short) {
    return "S_" + value;
} else if (value instanceof String) {
    return "s_" + value;
} else if (value instanceof ObjectRef) {
    return "R_" + ((ObjectRef) value).getEncoded(classNames);
} else {
    throw new IllegalArgumentException("unknown value");
```

```
        }
    }

    public static Object decode(String str, BiMap<String, String> classNames) {
        if (str == null) {
            return null;
        }
        String prefix = str.substring(0, 1);
        String value = str.substring(2);
        if (!"{".equals(prefix)) {
            String[] parts = str.split("_", 2);
            prefix = parts[0];
            value = parts[1];
        }
        switch (prefix) {
            case "{":
                Map<String, String> map = toObject(str, Map.class);
                Map<String, Object> objectMap = new
LinkedHashMap<>(hashMapInitialCapacity(map.size()));
                map.forEach((k, v) -> {
                    objectMap.put(k, decode(v, classNames));
                });
                return objectMap;
            case "I":
                return Integer.valueOf(value).intValue();
            case "J":
                return Long.valueOf(value).longValue();
            case "F":
                return Float.valueOf(value).floatValue();
            case "D":
                return Double.valueOf(value).doubleValue();
            case "Z":
                return Boolean.valueOf(value).booleanValue();
            case "B":
                return Byte.valueOf(value).byteValue();
            case "C":
                return value.charAt(0);
            case "S":
                return Short.valueOf(value).shortValue();
            case "s":
                return value;
            case "R":
```

```
                return new ObjectRef(value, classNames);
            case "[I":
                return decodeArray(value, Integer.TYPE, classNames);
            case "[J":
                return decodeArray(value, Long.TYPE, classNames);
            case "[F":
                return decodeArray(value, Float.TYPE, classNames);
            case "[D":
                return decodeArray(value, Double.TYPE, classNames);
            case "[Z":
                return decodeArray(value, Boolean.TYPE, classNames);
            case "[B":
                return decodeArray(value, Byte.TYPE, classNames);
            case "[C":
                return decodeArray(value, Character.TYPE, classNames);
            case "[S":
                return decodeArray(value, Short.TYPE, classNames);
            case "[R":
                return decodeArray(value, ObjectRef.class, classNames);
            default:
                throw new IllegalArgumentException("unknown string");
        }
    }

    public static byte[] compress(String data) {
//        try {
//            ByteArrayOutputStream bos = new ByteArrayOutputStream(data.length());
//            GZIPOutputStream gzip = new GZIPOutputStream(bos);
//            gzip.write(data.getBytes());
//            gzip.close();
//            byte[] compressed = bos.toByteArray();
//            bos.close();
//            return compressed;
//        } catch (IOException e) {
//            throw new RuntimeException(e);
//        }
        return data.getBytes();
    }

    public static String decompress(byte[] compressed) {
//        try {
//            ByteArrayInputStream bis = new ByteArrayInputStream(compressed);
```

```
//            GZIPInputStream gis = new GZIPInputStream(bis);
//            byte[] bytes = IOUtils.toByteArray(gis);
//            return new String(bytes);
//        } catch (IOException e) {
//            throw new RuntimeException(e);
//        }
        return new String(compressed);
    }

}

114:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\util\Log.java
 */
package io.nuls.contract.vm.util;

import io.nuls.contract.vm.OpCode;
import io.nuls.contract.vm.Result;
import io.nuls.contract.vm.code.MethodCode;

import java.util.Arrays;

public class Log {

    // TODO: 2018/4/27

    public static void loadClass(String className) {
        String log = "load class: " + className;
        log(log);
    }

    public static void runMethod(MethodCode methodCode) {
        String log = "run method: " + methodCode.className + "." + methodCode.name + " " +
methodCode.desc;
        log(log);
    }

    public static void continueMethod(MethodCode methodCode) {
        String log = "continue method: " + methodCode.className + "." + methodCode.name + " " +
methodCode.desc;
        log(log);
    }
```

```
    public static void endMethod(MethodCode methodCode) {
        String log = "end method: " + methodCode.className + "." + methodCode.name + " " +
methodCode.desc;
        log(log);
    }

    public static void nativeMethod(MethodCode methodCode) {
        String log = "native method: " + methodCode.className + "." + methodCode.name + " " +
methodCode.desc;
        log(log);
    }

    public static void nativeMethodResult(Result result) {
        String log = "native method result: " + result;
        log(log);
    }

    public static void opcode(OpCode opCode, Object... args) {
        String log = opCode.name();
        if (args != null && args.length > 0) {
            log += " " + Arrays.toString(args);
        }
        log(log);
    }

    public static void result(OpCode opCode, Object result, Object... args) {
        String log = opCode + " " + Arrays.toString(args) + " " + result;
        log(log);
    }

    public static void log(String log) {
        //System.out.println(log);
    }

}

115:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\util\Utils.java
 */
package io.nuls.contract.vm.util;
```

```
public class Utils {

    public static int arrayListInitialCapacity(int size) {
        return Math.max(size, 10);
    }

    public static int hashMapInitialCapacity(int size) {
        return Math.max((int) (size / 0.75) + 1, 16);
    }

}

116:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\VM.java
 */
package io.nuls.contract.vm;

import io.nuls.contract.entity.BlockHeaderDto;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.exception.ErrorException;
import io.nuls.contract.vm.instructions.comparisons.*;
import io.nuls.contract.vm.instructions.constants.Ldc;
import io.nuls.contract.vm.instructions.control.*;
import io.nuls.contract.vm.instructions.conversions.D2x;
import io.nuls.contract.vm.instructions.conversions.F2x;
import io.nuls.contract.vm.instructions.conversions.I2x;
import io.nuls.contract.vm.instructions.conversions.L2x;
import io.nuls.contract.vm.instructions.extended.Ifnonnull;
import io.nuls.contract.vm.instructions.extended.Ifnull;
import io.nuls.contract.vm.instructions.extended.Multianewarray;
import io.nuls.contract.vm.instructions.loads.*;
import io.nuls.contract.vm.instructions.math.*;
import io.nuls.contract.vm.instructions.references.*;
import io.nuls.contract.vm.instructions.stack.Dup;
import io.nuls.contract.vm.instructions.stack.Pop;
import io.nuls.contract.vm.instructions.stack.Swap;
import io.nuls.contract.vm.instructions.stores.*;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeAddress;
import io.nuls.contract.vm.program.ProgramExecutor;
import io.nuls.contract.vm.program.ProgramMethodArg;
```

```
import io.nuls.contract.vm.program.ProgramTransfer;
import io.nuls.contract.vm.program.impl.ProgramContext;
import io.nuls.contract.vm.program.impl.ProgramInvoke;
import io.nuls.contract.vm.util.Log;
import org.apache.commons.lang3.StringUtils;
import org.ethereum.core.Repository;
import org.objectweb.asm.tree.LookupSwitchInsnNode;
import org.objectweb.asm.tree.MultiANewArrayInsnNode;
import org.objectweb.asm.tree.TableSwitchInsnNode;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.lang.reflect.Array;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

public class VM {

    private Logger log = LoggerFactory.getLogger(VM.class);

    private static final int VM_STACK_MAX_SIZE = 512;

    public static final int MAX_GAS = 1000_0000;

    public final VMStack vmStack;

    public final Heap heap;

    public final MethodArea methodArea;

    private Result result;

    private Object resultValue;

    private VMContext vmContext;

    private ProgramInvoke programInvoke;

    private ProgramContext programContext;

    private ProgramExecutor programExecutor;
```

```
private Repository repository;

private long gasUsed;

private long gas;

private long startTime;

private long endTime;

private long elapsedTime;

private List<ProgramTransfer> transfers = new ArrayList<>();

private List<String> events = new ArrayList<>();

public VM() {
    this.vmStack = new VMStack(VM_STACK_MAX_SIZE);
    this.heap = new Heap(BigInteger.ZERO);
    this.heap.setVm(this);
    this.methodArea = new MethodArea();
    this.methodArea.setVm(this);
    this.result = new Result();
}

public VM(VM vm) {
    this.vmStack = new VMStack(VM_STACK_MAX_SIZE);
    this.heap = new Heap(vm.heap.getObjectRefCount());
    this.heap.setVm(this);
    this.methodArea = new MethodArea();
    this.methodArea.setVm(this);
    this.result = new Result();
}

public VM(Heap heap, MethodArea methodArea) {
    this.vmStack = new VMStack(VM_STACK_MAX_SIZE);
    this.heap = heap;
    this.heap.setVm(this);
    this.methodArea = methodArea;
    this.methodArea.setVm(this);
    this.result = new Result();
```

```
    }

    public boolean isEmptyFrame() {
        return this.vmStack.isEmpty();
    }

    public boolean isNotEmptyFrame() {
        return !isEmptyFrame();
    }

    public Frame lastFrame() {
        return this.vmStack.lastElement();
    }

    public void popFrame() {
        this.vmStack.pop();
    }

    public void endTime() {
        this.endTime = System.currentTimeMillis();
        this.elapsedTime = this.endTime - this.startTime;
    }

    public void initProgramContext(ProgramInvoke programInvoke) {
        this.programInvoke = programInvoke;

        programContext = new ProgramContext();
        programContext.setAddress(this.heap.newAddress(programInvoke.getAddress()));
        if (programInvoke.getSender() != null) {
programContext.setSender(this.heap.newAddress(NativeAddress.toString(programInvoke.getSen
der())));
        }
        programContext.setGasPrice(programInvoke.getPrice());
        programContext.setGas(programInvoke.getGasLimit());
        programContext.setValue(this.heap.newBigInteger(programInvoke.getValue().toString()));
        programContext.setNumber(programInvoke.getNumber());
        programContext.setEstimateGas(programInvoke.isEstimateGas());
    }

    private static final String CLASS_NAME = "java/util/HashMap";
    private static final String METHOD_NAME = "resize";
    private static final String METHOD_DESC = "()[Ljava/util/HashMap$Node;";
```

```
    public void run(MethodCode methodCode, Object[] args, boolean pushResult) {
        Frame frame = new Frame(this, methodCode, args);
        if (methodCode.isMethod(CLASS_NAME, METHOD_NAME, METHOD_DESC)) {
            frame.setAddGas(false);
        }
        this.vmStack.push(frame);
        run(pushResult);
        if (!frame.addGas) {
            frame.setAddGas(true);
        }
    }

    public void run(ObjectRef objectRef, MethodCode methodCode, VMContext vmContext,
ProgramInvoke programInvoke) {
        this.vmContext = vmContext;
        Object[] runArgs = runArgs(objectRef, methodCode, programInvoke.getArgs());
        if (isEnd()) {
            return;
        }
        initProgramContext(programInvoke);
        run(methodCode, runArgs, true);
    }

    private Object[] runArgs(ObjectRef objectRef, MethodCode methodCode, String[][] args) {
        final List runArgs = new ArrayList();
        runArgs.add(objectRef);
        final List<VariableType> argsVariableType = methodCode.argsVariableType;
        for (int i = 0; i < argsVariableType.size(); i++) {
            final VariableType variableType = argsVariableType.get(i);
            final ProgramMethodArg programMethodArg = methodCode.args.get(i);
            final String[] arg = args[i];
            String realArg = null;
            if (arg != null && arg.length > 0) {
                realArg = arg[0];
            }
            if (programMethodArg.isRequired()) {
                if (arg == null || arg.length < 1 || (!variableType.isArray() &&
StringUtils.isEmpty(realArg))) {
                    throw new RuntimeException(String.format("parameter %s required",
programMethodArg.getName()));
                }
```

```
        }
        if (arg == null || arg.length == 0) {
            runArgs.add(null);
        } else if (variableType.isArray()) {
            if (arg.length < 1) {
                runArgs.add(null);
            } else if (variableType.isPrimitiveType()) {
                Object array =
Array.newInstance(variableType.getComponentType().getPrimitiveTypeClass(), arg.length);
                for (int j = 0; j < arg.length; j++) {
                    String item = arg[j];
                    Object value = variableType.getComponentType().getPrimitiveValue(item);
                    Array.set(array, j, value);
                }
                final ObjectRef ref = this.heap.newArray(array, variableType, arg.length);
                runArgs.add(ref);
            } else if (variableType.getComponentType().isWrapperType()) {
                ObjectRef arrayRef = this.heap.newArray(variableType, arg.length);
                for (int j = 0; j < arg.length; j++) {
                    String item = arg[j];
                    if (item == null) {
                        continue;
                    }
                    ObjectRef ref;
                    if
(VariableType.CHAR_WRAPPER_TYPE.equals(variableType.getComponentType())) {
                        ref = this.heap.newCharacter(item.charAt(0));
                    } else {
                        ref = this.heap.runNewObject(variableType.getComponentType(), item);
                    }
                    if (isEnd()) {
                        return null;
                    }
                    this.heap.putArray(arrayRef, j, ref);
                }
                runArgs.add(arrayRef);
            } else {
                ObjectRef arrayRef = this.heap.newArray(VariableType.STRING_ARRAY_TYPE,
arg.length);
                for (int j = 0; j < arg.length; j++) {
                    String item = arg[j];
                    ObjectRef ref = this.heap.newString(item);
```

```
                this.heap.putArray(arrayRef, j, ref);
            }
            runArgs.add(arrayRef);
        }
    } else if (variableType.isPrimitive()) {
        final Object primitiveValue = variableType.getPrimitiveValue(realArg);
        runArgs.add(primitiveValue);
        if (variableType.isLong() || variableType.isDouble()) {
            runArgs.add(null);
        }
    } else if (VariableType.STRING_TYPE.equals(variableType)) {
        final ObjectRef ref = this.heap.newString(realArg);
        runArgs.add(ref);
    } else {
        final ObjectRef ref = this.heap.runNewObject(variableType, realArg);
        if (isEnd()) {
            return null;
        }
        runArgs.add(ref);
    }
  }
  return runArgs.toArray();
}

public void run(boolean pushResult) {
  if (this.startTime < 1) {
    this.startTime = System.currentTimeMillis();
  }
  if (this.result.isError()) {
    endTime();
    return;
  }
  if (!this.vmStack.isEmpty()) {
    final Frame frame = this.vmStack.lastElement();
    //Log.runMethod(frame.methodCode);
    while (frame.getCurrentInsnNode() != null && !frame.result.isEnded()) {
        step(frame);
        frame.step();
        if (isEnd()) {
            return;
        }
        if (frame != this.vmStack.lastElement()) {
```

```
                endTime();
                return;
            }
        }
        this.popFrame();
        //Log.endMethod(frame.methodCode);
        this.resultValue = frame.result.getValue();
        if (!this.vmStack.isEmpty()) {
            final Frame lastFrame = this.vmStack.lastElement();
            if (frame.result.getVariableType().isNotVoid()) {
                if (pushResult) {
                    lastFrame.operandStack.push(frame.result.getValue(),
frame.result.getVariableType());
                }
            }
            //Log.continueMethod(lastFrame.methodCode);
        } else {
            this.result = frame.result;
        }
    }
    endTime();
  }

  private boolean isEnd() {
    if (this.result.isError()) {
        endTime();
        return true;
    }
    if (this.result.isException()) {
        endTime();
        return true;
    }
    return false;
  }

  private void step(Frame frame) {

    OpCode opCode = frame.currentOpCode();

    if (opCode == null) {
        if (frame.getCurrentInsnNode() != null && frame.getCurrentInsnNode().getOpcode() >= 0) {
            frame.nonsupportOpCode();
```

```
    }
    return;
}

if (frame.addGas) {
    int gasCost = gasCost(frame, opCode);
    addGasUsed(gasCost);
}

switch (opCode) {
    case NOP:
        //Nop.nop(frame);
        break;
    case ACONST_NULL:
        //Aconst.aconst_null(frame);
        frame.operandStack.pushRef(null);
        break;
    case ICONST_M1:
        //Iconst.iconst_m1(frame);
        frame.operandStack.pushInt(-1);
        break;
    case ICONST_0:
        //Iconst.iconst_0(frame);
        frame.operandStack.pushInt(0);
        break;
    case ICONST_1:
        //Iconst.iconst_1(frame);
        frame.operandStack.pushInt(1);
        break;
    case ICONST_2:
        //Iconst.iconst_2(frame);
        frame.operandStack.pushInt(2);
        break;
    case ICONST_3:
        //Iconst.iconst_3(frame);
        frame.operandStack.pushInt(3);
        break;
    case ICONST_4:
        //Iconst.iconst_4(frame);
        frame.operandStack.pushInt(4);
        break;
    case ICONST_5:
```

```
    //Iconst.iconst_5(frame);
    frame.operandStack.pushInt(5);
    break;
case LCONST_0:
    //Lconst.lconst_0(frame);
    frame.operandStack.pushLong(0L);
    break;
case LCONST_1:
    //Lconst.lconst_1(frame);
    frame.operandStack.pushLong(1L);
    break;
case FCONST_0:
    //Fconst.fconst_0(frame);
    frame.operandStack.pushFloat(0.0F);
    break;
case FCONST_1:
    //Fconst.fconst_1(frame);
    frame.operandStack.pushFloat(1.0F);
    break;
case FCONST_2:
    //Fconst.fconst_2(frame);
    frame.operandStack.pushFloat(2.0F);
    break;
case DCONST_0:
    //Dconst.dconst_0(frame);
    frame.operandStack.pushDouble(0.0D);
    break;
case DCONST_1:
    //Dconst.dconst_1(frame);
    frame.operandStack.pushDouble(1.0D);
    break;
case BIPUSH:
    //Xipush.bipush(frame);
    frame.operandStack.pushInt(frame.intInsnNode().operand);
    break;
case SIPUSH:
    //Xipush.sipush(frame);
    frame.operandStack.pushInt(frame.intInsnNode().operand);
    break;
case LDC:
    Ldc.ldc(frame);
    break;
```

```
case ILOAD:
    Iload.iload(frame);
    break;
case LLOAD:
    Lload.lload(frame);
    break;
case FLOAD:
    Fload.fload(frame);
    break;
case DLOAD:
    Dload.dload(frame);
    break;
case ALOAD:
    Aload.aload(frame);
    break;
case IALOAD:
    Xaload.iaload(frame);
    break;
case LALOAD:
    Xaload.laload(frame);
    break;
case FALOAD:
    Xaload.faload(frame);
    break;
case DALOAD:
    Xaload.daload(frame);
    break;
case AALOAD:
    Xaload.aaload(frame);
    break;
case BALOAD:
    Xaload.baload(frame);
    break;
case CALOAD:
    Xaload.caload(frame);
    break;
case SALOAD:
    Xaload.saload(frame);
    break;
case ISTORE:
    Istore.istore(frame);
    break;
```

```
case LSTORE:
    Lstore.lstore(frame);
    break;
case FSTORE:
    Fstore.fstore(frame);
    break;
case DSTORE:
    Dstore.dstore(frame);
    break;
case ASTORE:
    Astore.astore(frame);
    break;
case IASTORE:
    Xastore.iastore(frame);
    break;
case LASTORE:
    Xastore.lastore(frame);
    break;
case FASTORE:
    Xastore.fastore(frame);
    break;
case DASTORE:
    Xastore.dastore(frame);
    break;
case AASTORE:
    Xastore.aastore(frame);
    break;
case BASTORE:
    Xastore.bastore(frame);
    break;
case CASTORE:
    Xastore.castore(frame);
    break;
case SASTORE:
    Xastore.sastore(frame);
    break;
case POP:
    Pop.pop(frame);
    break;
case POP2:
    Pop.pop2(frame);
    break;
```

```
case DUP:
    Dup.dup(frame);
    break;
case DUP_X1:
    Dup.dup_x1(frame);
    break;
case DUP_X2:
    Dup.dup_x2(frame);
    break;
case DUP2:
    Dup.dup2(frame);
    break;
case DUP2_X1:
    Dup.dup2_x1(frame);
    break;
case DUP2_X2:
    Dup.dup2_x2(frame);
    break;
case SWAP:
    Swap.swap(frame);
    break;
case IADD:
    Add.iadd(frame);
    break;
case LADD:
    Add.ladd(frame);
    break;
case FADD:
    Add.fadd(frame);
    break;
case DADD:
    Add.dadd(frame);
    break;
case ISUB:
    Sub.isub(frame);
    break;
case LSUB:
    Sub.lsub(frame);
    break;
case FSUB:
    Sub.fsub(frame);
    break;
```

```
case DSUB:
    Sub.dsub(frame);
    break;
case IMUL:
    Mul.imul(frame);
    break;
case LMUL:
    Mul.lmul(frame);
    break;
case FMUL:
    Mul.fmul(frame);
    break;
case DMUL:
    Mul.dmul(frame);
    break;
case IDIV:
    Div.idiv(frame);
    break;
case LDIV:
    Div.ldiv(frame);
    break;
case FDIV:
    Div.fdiv(frame);
    break;
case DDIV:
    Div.ddiv(frame);
    break;
case IREM:
    Rem.irem(frame);
    break;
case LREM:
    Rem.lrem(frame);
    break;
case FREM:
    Rem.frem(frame);
    break;
case DREM:
    Rem.drem(frame);
    break;
case INEG:
    Neg.ineg(frame);
    break;
```

```
case LNEG:
    Neg.lneg(frame);
    break;
case FNEG:
    Neg.fneg(frame);
    break;
case DNEG:
    Neg.dneg(frame);
    break;
case ISHL:
    Shl.ishl(frame);
    break;
case LSHL:
    Shl.lshl(frame);
    break;
case ISHR:
    Shr.ishr(frame);
    break;
case LSHR:
    Shr.lshr(frame);
    break;
case IUSHR:
    Ushr.iushr(frame);
    break;
case LUSHR:
    Ushr.lushr(frame);
    break;
case IAND:
    And.iand(frame);
    break;
case LAND:
    And.land(frame);
    break;
case IOR:
    Or.ior(frame);
    break;
case LOR:
    Or.lor(frame);
    break;
case IXOR:
    Xor.ixor(frame);
    break;
```

```
case LXOR:
    Xor.lxor(frame);
    break;
case IINC:
    Iinc.iinc(frame);
    break;
case I2L:
    I2x.i2l(frame);
    break;
case I2F:
    I2x.i2f(frame);
    break;
case I2D:
    I2x.i2d(frame);
    break;
case L2I:
    L2x.l2i(frame);
    break;
case L2F:
    L2x.l2f(frame);
    break;
case L2D:
    L2x.l2d(frame);
    break;
case F2I:
    F2x.f2i(frame);
    break;
case F2L:
    F2x.f2l(frame);
    break;
case F2D:
    F2x.f2d(frame);
    break;
case D2I:
    D2x.d2i(frame);
    break;
case D2L:
    D2x.d2l(frame);
    break;
case D2F:
    D2x.d2f(frame);
    break;
```

```
case I2B:
    I2x.i2b(frame);
    break;
case I2C:
    I2x.i2c(frame);
    break;
case I2S:
    I2x.i2s(frame);
    break;
case LCMP:
    Lcmp.lcmp(frame);
    break;
case FCMPL:
    Fcmp.fcmpl(frame);
    break;
case FCMPG:
    Fcmp.fcmpg(frame);
    break;
case DCMPL:
    Dcmp.dcmpl(frame);
    break;
case DCMPG:
    Dcmp.dcmpg(frame);
    break;
case IFEQ:
    IfCmp.ifeq(frame);
    break;
case IFNE:
    IfCmp.ifne(frame);
    break;
case IFLT:
    IfCmp.iflt(frame);
    break;
case IFGE:
    IfCmp.ifge(frame);
    break;
case IFGT:
    IfCmp.ifgt(frame);
    break;
case IFLE:
    IfCmp.ifle(frame);
    break;
```

```
case IF_ICMPEQ:
    IfIcmp.if_icmpeq(frame);
    break;
case IF_ICMPNE:
    IfIcmp.if_icmpne(frame);
    break;
case IF_ICMPLT:
    IfIcmp.if_icmplt(frame);
    break;
case IF_ICMPGE:
    IfIcmp.if_icmpge(frame);
    break;
case IF_ICMPGT:
    IfIcmp.if_icmpgt(frame);
    break;
case IF_ICMPLE:
    IfIcmp.if_icmple(frame);
    break;
case IF_ACMPEQ:
    IfAcmp.if_acmpeq(frame);
    break;
case IF_ACMPNE:
    IfAcmp.if_acmpne(frame);
    break;
case GOTO:
    Goto.goto_(frame);
    break;
case JSR:
    Jsr.jsr(frame);
    break;
case RET:
    Ret.ret(frame);
    break;
case TABLESWITCH:
    Tableswitch.tableswitch(frame);
    break;
case LOOKUPSWITCH:
    Lookupswitch.lookupswitch(frame);
    break;
case IRETURN:
    Return.ireturn(frame);
    break;
```

```
case LRETURN:
    Return.lreturn(frame);
    break;
case FRETURN:
    Return.freturn(frame);
    break;
case DRETURN:
    Return.dreturn(frame);
    break;
case ARETURN:
    Return.areturn(frame);
    break;
case RETURN:
    Return.return_(frame);
    break;
case GETSTATIC:
    Getstatic.getstatic(frame);
    break;
case PUTSTATIC:
    Putstatic.putstatic(frame);
    break;
case GETFIELD:
    Getfield.getfield(frame);
    break;
case PUTFIELD:
    Putfield.putfield(frame);
    break;
case INVOKEVIRTUAL:
    Invokevirtual.invokevirtual(frame);
    break;
case INVOKESPECIAL:
    Invokespecial.invokespecial(frame);
    break;
case INVOKESTATIC:
    Invokestatic.invokestatic(frame);
    break;
case INVOKEINTERFACE:
    Invokeinterface.invokeinterface(frame);
    break;
case INVOKEDYNAMIC:
    Invokedynamic.invokedynamic(frame);
    break;
```

```
        case NEW:
            New.new_(frame);
            break;
        case NEWARRAY:
            Newarray.newarray(frame);
            break;
        case ANEWARRAY:
            Anewarray.anewarray(frame);
            break;
        case ARRAYLENGTH:
            Arraylength.arraylength(frame);
            break;
        case ATHROW:
            Athrow.athrow(frame);
            break;
        case CHECKCAST:
            Checkcast.checkcast(frame);
            break;
        case INSTANCEOF:
            Instanceof.instanceof_(frame);
            break;
        case MONITORENTER:
            Monitorenter.monitorenter(frame);
            break;
        case MONITOREXIT:
            Monitorexit.monitorexit(frame);
            break;
        case MULTIANEWARRAY:
            Multianewarray.multianewarray(frame);
            break;
        case IFNULL:
            Ifnull.ifnull(frame);
            break;
        case IFNONNULL:
            Ifnonnull.ifnonnull(frame);
            break;
        default:
            frame.nonsupportOpCode();
            break;
    }
}
```

```
public int gasCost(Frame frame, OpCode opCode) {
    int gasCost = 1;
    switch (opCode) {
        case NOP:
            break;
        case ACONST_NULL:
        case ICONST_M1:
        case ICONST_0:
        case ICONST_1:
        case ICONST_2:
        case ICONST_3:
        case ICONST_4:
        case ICONST_5:
        case LCONST_0:
        case LCONST_1:
        case FCONST_0:
        case FCONST_1:
        case FCONST_2:
        case DCONST_0:
        case DCONST_1:
        case BIPUSH:
        case SIPUSH:
            gasCost = GasCost.CONSTANT;
            break;
        case LDC:
            Object value = frame.ldcInsnNode().cst;
            if (value instanceof Number) {
                gasCost = GasCost.LDC;
            } else {
                gasCost = Math.max(value.toString().length(), 1) * GasCost.LDC;
            }
            break;
        case ILOAD:
        case LLOAD:
        case FLOAD:
        case DLOAD:
        case ALOAD:
            gasCost = GasCost.LOAD;
            break;
        case IALOAD:
        case LALOAD:
        case FALOAD:
```

```
case DALOAD:
case AALOAD:
case BALOAD:
case CALOAD:
case SALOAD:
    gasCost = GasCost.ARRAYLOAD;
    break;
case ISTORE:
case LSTORE:
case FSTORE:
case DSTORE:
case ASTORE:
    gasCost = GasCost.STORE;
    break;
case IASTORE:
case LASTORE:
case FASTORE:
case DASTORE:
case AASTORE:
case BASTORE:
case CASTORE:
case SASTORE:
    gasCost = GasCost.ARRAYSTORE;
    break;
case POP:
case POP2:
case DUP:
case DUP_X1:
case DUP_X2:
case DUP2:
case DUP2_X1:
case DUP2_X2:
case SWAP:
    gasCost = GasCost.STACK;
    break;
case IADD:
case LADD:
case FADD:
case DADD:
case ISUB:
case LSUB:
case FSUB:
```

```
case DSUB:
case IMUL:
case LMUL:
case FMUL:
case DMUL:
case IDIV:
case LDIV:
case FDIV:
case DDIV:
case IREM:
case LREM:
case FREM:
case DREM:
case INEG:
case LNEG:
case FNEG:
case DNEG:
case ISHL:
case LSHL:
case ISHR:
case LSHR:
case IUSHR:
case LUSHR:
case IAND:
case LAND:
case IOR:
case LOR:
case IXOR:
case LXOR:
case IINC:
    gasCost = GasCost.MATH;
    break;
case I2L:
case I2F:
case I2D:
case L2I:
case L2F:
case L2D:
case F2I:
case F2L:
case F2D:
case D2I:
```

```
case D2L:
case D2F:
case I2B:
case I2C:
case I2S:
    gasCost = GasCost.CONVERSION;
    break;
case LCMP:
case FCMPL:
case FCMPG:
case DCMPL:
case DCMPG:
case IFEQ:
case IFNE:
case IFLT:
case IFGE:
case IFGT:
case IFLE:
case IF_ICMPEQ:
case IF_ICMPNE:
case IF_ICMPLT:
case IF_ICMPGE:
case IF_ICMPGT:
case IF_ICMPLE:
case IF_ACMPEQ:
case IF_ACMPNE:
    gasCost = GasCost.COMPARISON;
    break;
case GOTO:
case JSR:
case RET:
    gasCost = GasCost.CONTROL;
    break;
case TABLESWITCH:
    TableSwitchInsnNode table = frame.tableSwitchInsnNode();
    gasCost = Math.max(table.max - table.min, 1) * GasCost.TABLESWITCH;
    break;
case LOOKUPSWITCH:
    LookupSwitchInsnNode lookup = frame.lookupSwitchInsnNode();
    gasCost = Math.max(lookup.keys.size(), 1) * GasCost.LOOKUPSWITCH;
    break;
case IRETURN:
```

```
            case LRETURN:
            case FRETURN:
            case DRETURN:
            case ARETURN:
            case RETURN:
                gasCost = GasCost.CONTROL;
                break;
            case GETSTATIC:
            case PUTSTATIC:
            case GETFIELD:
            case PUTFIELD:
            case INVOKEVIRTUAL:
            case INVOKESPECIAL:
            case INVOKESTATIC:
            case INVOKEINTERFACE:
            case INVOKEDYNAMIC:
            case NEW:
                gasCost = GasCost.REFERENCE;
                break;
            case NEWARRAY:
            case ANEWARRAY:
                int count = frame.operandStack.popInt();
                gasCost = Math.max(count, 1) * GasCost.NEWARRAY;
                frame.operandStack.pushInt(count);
                break;
            case ARRAYLENGTH:
            case ATHROW:
            case CHECKCAST:
            case INSTANCEOF:
            case MONITORENTER:
            case MONITOREXIT:
                gasCost = GasCost.REFERENCE;
                break;
            case MULTIANEWARRAY:
                MultiANewArrayInsnNode multiANewArrayInsnNode =
frame.multiANewArrayInsnNode();
                int size = 1;
                int[] dimensions = new int[multiANewArrayInsnNode.dims];
                for (int i = multiANewArrayInsnNode.dims - 1; i >= 0; i--) {
                    int length = frame.operandStack.popInt();
                    if (length > 0) {
                        size *= length;
```

```
                }
                dimensions[i] = length;
            }
            for (int dimension : dimensions) {
                frame.operandStack.pushInt(dimension);
            }
            gasCost = size * GasCost.MULTIANEWARRAY;
            break;
        case IFNULL:
        case IFNONNULL:
            gasCost = GasCost.EXTENDED;
            break;
        default:
            break;
    }
    return gasCost;
}

public BlockHeaderDto getBlockHeader(long number) {
    if (this.vmContext != null) {
        BlockHeaderDto blockHeader = null;
        try {
            if (number == programInvoke.getNumber() + 1) {
                blockHeader = this.vmContext.getCurrentBlockHeader();
            } else {
                blockHeader = this.vmContext.getBlockHeader(number);
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        if (blockHeader == null) {
            log.error(String.format("blockHeader is null, number: %s", number));
        }
        return blockHeader;
    } else {
        throw new RuntimeException(String.format("vmContext is null, number: %s", number));
    }
}

//    public BlockHeaderDto getBlockHeader(long number) {
//        BlockHeaderDto blockHeaderDto = new BlockHeaderDto();
//        blockHeaderDto.setHash("hash" + number);
```

```
//        blockHeaderDto.setHeight(number);
//        blockHeaderDto.setTxCount(100);
//
blockHeaderDto.setPackingAddress(AddressTool.getAddress("TTapY7gpBm1DHEgwguSFFtuK3
JvGZVKK"));
//        blockHeaderDto.setTime(1535012808001L);
//        return blockHeaderDto;
//    }

    public Result getResult() {
        return result;
    }

    public String getResultString() {
        String result = null;
        Object resultValue = getResult().getValue();
        if (resultValue != null) {
            if (resultValue instanceof ObjectRef) {
                if (getResult().isError() || getResult().isException()) {
                    setResult(new Result());
                }
                result = this.heap.runToString((ObjectRef) resultValue);
            } else {
                result = resultValue.toString();
            }
        }
        return result;
    }

    public Object getResultValue() {
        return resultValue;
    }

    public VMContext getVmContext() {
        return vmContext;
    }

    public ProgramInvoke getProgramInvoke() {
        return programInvoke;
    }

    public ProgramContext getProgramContext() {
```

```
    return programContext;
}

public ProgramExecutor getProgramExecutor() {
    return programExecutor;
}

public long getGasUsed() {
    return gasUsed;
}

public long getGas() {
    return gas;
}

public long getGasLeft() {
    return gas - gasUsed;
}

public long getStartTime() {
    return startTime;
}

public long getEndTime() {
    return endTime;
}

public long getElapsedTime() {
    return elapsedTime;
}

public List<ProgramTransfer> getTransfers() {
    return transfers;
}

public List<String> getEvents() {
    return events;
}

public void setResult(Result result) {
    this.result = result;
}
```

```
    public void setProgramExecutor(ProgramExecutor programExecutor) {
        this.programExecutor = programExecutor;
    }

    public Repository getRepository() {
        return repository;
    }

    public void setRepository(Repository repository) {
        this.repository = repository;
    }

    public void addGasUsed(long needGas) {
        long gasUsed = this.gasUsed + needGas;
        if (this.gas > 0 && gasUsed > this.gas) {
            this.gasUsed = this.gas;
            throw new ErrorException("not enough gas", this.gasUsed, null);
        } else {
            this.gasUsed = gasUsed;
        }
    }

    public void setGas(long gas) {
        this.gas = gas;
    }

}

117:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\VMFactory.java
 */
package io.nuls.contract.vm;

import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.ClassCodeLoader;
import io.nuls.contract.vm.program.impl.ProgramConstants;
import org.apache.commons.lang3.ArrayUtils;

import java.util.LinkedHashMap;
import java.util.Map;
```

```
public class VMFactory {

    public static final Map<String, ClassCode> VM_INIT_CLASS_CODES = new
LinkedHashMap(1024);

    public static final VM VM;

    private static final String[] IGNORE_CLINIT = new String[]{
            "java/io/File",
            "java/io/FileDescriptor",
            "java/io/ObjectInputStream",
            "java/io/ObjectOutputStream",
            "java/io/ObjectStreamClass",
            "java/io/ObjectStreamClass$FieldReflector",
            "java/lang/SecurityManager",
            "java/lang/invoke/BoundMethodHandle",
            "java/lang/invoke/BoundMethodHandle$SpeciesData",
            "java/lang/invoke/DirectMethodHandle",
            "java/lang/invoke/Invokers",
            "java/lang/invoke/LambdaForm",
            "java/lang/invoke/LambdaForm$BasicType",
            "java/lang/invoke/LambdaForm$NamedFunction",
            "java/lang/invoke/MethodHandle",
            "java/lang/invoke/MethodHandles$Lookup",
            "java/lang/invoke/MethodType",
            "java/lang/ref/Reference",
            "java/lang/reflect/AccessibleObject",
            "java/net/InetAddress",
            "java/net/NetworkInterface",
            "java/nio/charset/Charset",
            "java/security/ProtectionDomain",
            "java/security/Provider",
            "java/time/OffsetTime",
            "java/time/ZoneOffset",
            "java/util/Locale",
            "java/util/Locale$1",
            "java/util/Random",
            "sun/misc/URLClassPath",
            "sun/security/util/Debug",
            "sun/util/locale/BaseLocale",
    };
```

```
    public static final String[] INIT_CLASS_CACHE = new String[]{
            "java/lang/Byte$ByteCache",
            "java/lang/Short$ShortCache",
            "java/lang/Character$CharacterCache",
            "java/lang/Integer$IntegerCache",
            "java/lang/Long$LongCache",
    };

    static {
        String[] classes = ArrayUtils.addAll(ProgramConstants.VM_INIT_CLASS_NAMES,
ProgramConstants.CONTRACT_USED_CLASS_NAMES);
        classes = ArrayUtils.addAll(classes,
ProgramConstants.CONTRACT_LAZY_USED_CLASS_NAMES);
        classes = ArrayUtils.addAll(classes, ProgramConstants.SDK_CLASS_NAMES);
        classes = ArrayUtils.addAll(classes, INIT_CLASS_CACHE);
        for (String className : classes) {
            VM_INIT_CLASS_CODES.put(className,
ClassCodeLoader.loadFromResource(className));
        }
        for (int i = 0; i < classes.length; i++) {
            VM_INIT_CLASS_CODES.putAll(ClassCodeLoader.loadAll(classes[i],
ClassCodeLoader::loadFromResource));
        }
        VM = newVM();
        MethodArea.INIT_CLASS_CODES.putAll(VM.methodArea.getClassCodes());
        MethodArea.INIT_METHOD_CODES.putAll(VM.methodArea.getMethodCodes());
        Heap.INIT_OBJECTS.putAll(VM.heap.objects);
        Heap.INIT_ARRAYS.putAll(VM.heap.arrays);
    }

    public static VM createVM() {
        return new VM(VM);
    }

    public static VM newVM() {
        VM vm = new VM();
        for (String key : VM_INIT_CLASS_CODES.keySet()) {
            ClassCode classCode = VM_INIT_CLASS_CODES.get(key);
            if (ArrayUtils.contains(IGNORE_CLINIT, key)) {
                continue;
            }
            //System.out.println(key);
```

```
            vm.methodArea.loadClassCode(classCode);
        }
        return vm;
    }

}
```

118:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\VMStack.java

```
 */
package io.nuls.contract.vm;

import java.util.Stack;

public class VMStack extends Stack<Frame> {

    private final int maxSize;

    public VMStack(int maxSize) {
        this.maxSize = maxSize;
    }

    @Override
    public Frame push(Frame frame) {
        if (size() > maxSize) {
            frame.throwStackOverflowError();
        }
        return super.push(frame);
    }

    @Override
    public synchronized Frame pop() {
        return super.pop();
    }

}
```

119:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\config\BlockchainNetConfig.java

```
 * Created by Anton Nashatyrev on 25.02.2016.
 */
public interface BlockchainNetConfig {
```

```
    /**
     * Get the config for the specific block
     */
    //BlockchainConfig getConfigForBlock(long blockNumber);

    /**
     * Returns the constants common for all the blocks in this blockchain
     */
    Constants getCommonConstants();
}
```

120:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\config\CommonConfig.java

```
import org.ethereum.datasource.leveldb.LevelDbDataSource;
import org.ethereum.db.DbFlushManager;
import org.ethereum.db.HeaderStore;
import org.ethereum.db.RepositoryRoot;
import org.ethereum.db.StateSource;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.HashSet;
import java.util.Set;

public class CommonConfig {
    private static final Logger logger = LoggerFactory.getLogger("general");
    private Set<DbSource> dbSources = new HashSet<>();

    private static CommonConfig defaultInstance;

    public static CommonConfig getDefault() {
        if (defaultInstance == null) {
            defaultInstance = new CommonConfig();
        }
        return defaultInstance;
    }

    public SystemProperties systemProperties() {
        return SystemProperties.getSpringDefault();
    }
```

```
    public Repository defaultRepository() {
        return new RepositoryRoot(stateSource(), null);
    }

    public Repository repository(byte[] stateRoot) {
        return new RepositoryRoot(stateSource(), stateRoot);
    }

    /**
     * A source of nodes for state trie and all contract storage tries. <br/>
     * This source provides contract code too. <br/><br/>
     * <p>
     * Picks node by 16-bytes prefix of its key. <br/>
     * Within {@link NodeKeyCompositor} this source is a part of ref counting workaround<br/><br/>
     *
     * <b>Note:</b> is eligible as a public node provider, like in {@link Eth63};
     * {@link StateSource} is intended for inner usage only
     *
     * @see NodeKeyCompositor
     * @see RepositoryRoot#RepositoryRoot(Source, byte[])
     * @see Eth63
     */
    private Source<byte[], byte[]> trieNodeSource;

    public Source<byte[], byte[]> trieNodeSource() {
        if (trieNodeSource == null) {
            DbSource<byte[]> db = blockchainDB();
            Source<byte[], byte[]> src = new PrefixLookupSource<>(db,
NodeKeyCompositor.PREFIX_BYTES);
            trieNodeSource = new XorDataSource<>(src, HashUtil.sha3("state".getBytes()));
        }
        return trieNodeSource;
    }

    private StateSource stateSource;

    public StateSource stateSource() {
        if (this.stateSource == null) {
            //fastSyncCleanUp();
            StateSource stateSource = new StateSource(blockchainSource("state"),
                    systemProperties().databasePruneDepth() >= 0);
```

```
            dbFlushManager().addCache(stateSource.getWriteCache());

            this.stateSource = stateSource;
        }
        return this.stateSource;
    }

    public Source<byte[], byte[]> cachedDbSource(String name) {
        AbstractCachedSource<byte[], byte[]> writeCache = new AsyncWriteCache<byte[],
byte[]>(blockchainSource(name)) {
            @Override
            protected WriteCache<byte[], byte[]> createCache(Source<byte[], byte[]> source) {
                WriteCache.BytesKey<byte[]> ret = new WriteCache.BytesKey<>(source,
WriteCache.CacheType.SIMPLE);
                ret.withSizeEstimators(MemSizeEstimator.ByteArrayEstimator,
MemSizeEstimator.ByteArrayEstimator);
                ret.setFlushSource(true);
                return ret;
            }
        }.withName(name);
        dbFlushManager().addCache(writeCache);
        return writeCache;
    }

    public Source<byte[], byte[]> blockchainSource(String name) {
        return new XorDataSource<>(blockchainDbCache(), HashUtil.sha3(name.getBytes()));
    }

    private AbstractCachedSource<byte[], byte[]> blockchainDbCache;

    public AbstractCachedSource<byte[], byte[]> blockchainDbCache() {
        if (blockchainDbCache == null) {
            WriteCache.BytesKey<byte[]> ret = new WriteCache.BytesKey<>(
                    new BatchSourceWriter<>(blockchainDB()), WriteCache.CacheType.SIMPLE);
            ret.setFlushSource(true);
            blockchainDbCache = ret;
        }
        return blockchainDbCache;
    }

    public DbSource<byte[]> keyValueDataSource(String name) {
        return keyValueDataSource(name, DbSettings.DEFAULT);
```

```
}

public DbSource<byte[]> keyValueDataSource(String name, DbSettings settings) {
    String dataSource = systemProperties().getKeyValueDataSource();
    try {
        DbSource<byte[]> dbSource;
        if ("inmem".equals(dataSource)) {
            dbSource = new HashMapDB<>();
        } else {
            dbSource = levelDbDataSource();
        }
        dbSource.setName(name);
        dbSource.init(settings);
        dbSources.add(dbSource);
        return dbSource;
    } finally {
        logger.debug(dataSource + " key-value data source created: " + name);
    }
}

protected LevelDbDataSource levelDbDataSource() {
    return new LevelDbDataSource();
}

private void resetDataSource(Source source) {
    if (source instanceof DbSource) {
        ((DbSource) source).reset();
    } else {
        throw new Error("Cannot cleanup non-db Source");
    }
}

private DbSource<byte[]> headerSource;

public DbSource<byte[]> headerSource() {
    if (headerSource == null) {
        headerSource = keyValueDataSource("headers");
    }
    return headerSource;
}

private HeaderStore headerStore;
```

```
    public HeaderStore headerStore() {
        if (this.headerStore == null) {
            DbSource<byte[]> dataSource = headerSource();

            WriteCache.BytesKey<byte[]> cache = new WriteCache.BytesKey<>(
                    new BatchSourceWriter<>(dataSource), WriteCache.CacheType.SIMPLE);
            cache.setFlushSource(true);
            dbFlushManager().addCache(cache);

            HeaderStore headerStore = new HeaderStore();
            Source<byte[], byte[]> headers = new XorDataSource<>(cache,
HashUtil.sha3("header".getBytes()));
            Source<byte[], byte[]> index = new XorDataSource<>(cache,
HashUtil.sha3("index".getBytes()));
            headerStore.init(index, headers);

            this.headerStore = headerStore;
        }
        return this.headerStore;
    }

    private DbSource<byte[]> blockchainDB;

    public DbSource<byte[]> blockchainDB() {
        if (blockchainDB == null) {
            DbSettings settings = DbSettings.newInstance()
                    .withMaxOpenFiles(systemProperties().getConfig().getInt("database.maxOpenFiles"))
                    .withMaxThreads(Math.max(1, Runtime.getRuntime().availableProcessors() / 2));

            blockchainDB = keyValueDataSource("blockchain", settings);
        }
        return blockchainDB;
    }

    private DbFlushManager dbFlushManager;

    public DbFlushManager dbFlushManager() {
        if (dbFlushManager == null) {
            dbFlushManager = new DbFlushManager(systemProperties(), dbSources,
blockchainDbCache());
        }
```

```
        return dbFlushManager;
    }

}

121:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\org\ethereum\config\Constants.java
/**
 * Describes different constants specific for a blockchain
 * <p>
 * Created by Anton Nashatyrev on 25.02.2016.
 */
public class Constants {
    private static final int MAXIMUM_EXTRA_DATA_SIZE = 32;
    private static final int MIN_GAS_LIMIT = 125000;
    private static final int GAS_LIMIT_BOUND_DIVISOR = 1024;
    private static final BigInteger MINIMUM_DIFFICULTY = BigInteger.valueOf(131072);
    private static final BigInteger DIFFICULTY_BOUND_DIVISOR = BigInteger.valueOf(2048);
    private static final int EXP_DIFFICULTY_PERIOD = 100000;

    private static final int UNCLE_GENERATION_LIMIT = 7;
    private static final int UNCLE_LIST_LIMIT = 2;

    private static final int BEST_NUMBER_DIFF_LIMIT = 100;

    private static final BigInteger BLOCK_REWARD = EtherUtil.convert(1500,
EtherUtil.Unit.FINNEY);  // 1.5 ETH

    private static final BigInteger SECP256K1N = new
BigInteger("fffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141", 16);

    private static final int LONGEST_CHAIN = 192;

    public int getDURATION_LIMIT() {
        return 8;
    }

    public BigInteger getInitialNonce() {
        return BigInteger.ZERO;
    }

    public int getMAXIMUM_EXTRA_DATA_SIZE() {
```

```
    return MAXIMUM_EXTRA_DATA_SIZE;
}

public int getMIN_GAS_LIMIT() {
    return MIN_GAS_LIMIT;
}

public int getGAS_LIMIT_BOUND_DIVISOR() {
    return GAS_LIMIT_BOUND_DIVISOR;
}

public BigInteger getMINIMUM_DIFFICULTY() {
    return MINIMUM_DIFFICULTY;
}

public BigInteger getDIFFICULTY_BOUND_DIVISOR() {
    return DIFFICULTY_BOUND_DIVISOR;
}

public int getEXP_DIFFICULTY_PERIOD() {
    return EXP_DIFFICULTY_PERIOD;
}

public int getUNCLE_GENERATION_LIMIT() {
    return UNCLE_GENERATION_LIMIT;
}

public int getUNCLE_LIST_LIMIT() {
    return UNCLE_LIST_LIMIT;
}

public int getBEST_NUMBER_DIFF_LIMIT() {
    return BEST_NUMBER_DIFF_LIMIT;
}

public BigInteger getBLOCK_REWARD() {
    return BLOCK_REWARD;
}

public int getMAX_CONTRACT_SZIE() {
    return Integer.MAX_VALUE;
}
```

```
    /**
     * Introduced in the Homestead release
     */
    public boolean createEmptyContractOnOOG() {
        return true;
    }

    /**
     * New DELEGATECALL opcode introduced in the Homestead release. Before Homestead this
opcode should generate
     * exception
     */
    public boolean hasDelegateCallOpcode() {
        return false;
    }

    /**
     * Introduced in the Homestead release
     */
    public static BigInteger getSECP256K1N() {
        return SECP256K1N;
    }

    public static int getLONGEST_CHAIN() {
        return LONGEST_CHAIN;
    }
}
```

122:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\config\DefaultConfig.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * @author Roman Mandeleil
 * Created on: 27/01/2015 01:05
 */
public class DefaultConfig {
    private static Logger logger = LoggerFactory.getLogger("general");

    CommonConfig commonConfig = CommonConfig.getDefault();
```

```
    SystemProperties config = SystemProperties.getDefault();

    private static DefaultConfig defaultInstance;

    public static DefaultConfig getDefault() {
        if (defaultInstance == null) {
            defaultInstance = new DefaultConfig();
        }
        return defaultInstance;
    }

    public DefaultConfig() {
        Thread.setDefaultUncaughtExceptionHandler((t, e) -> logger.error("Uncaught exception", e));
    }

    private BlockStore blockStore;

    public BlockStore blockStore() {
        if (blockStore == null) {
            //commonConfig.fastSyncCleanUp();
            IndexedBlockStore indexedBlockStore = new IndexedBlockStore();
            Source<byte[], byte[]> block = commonConfig.cachedDbSource("block");
            Source<byte[], byte[]> index = commonConfig.cachedDbSource("index");
            indexedBlockStore.init(index, block);
            blockStore = indexedBlockStore;
        }
        return blockStore;
    }

    private PruneManager pruneManager;

    public PruneManager pruneManager() {
        if (pruneManager == null) {
            if (config.databasePruneDepth() >= 0) {
                pruneManager = new PruneManager((IndexedBlockStore) blockStore(),
commonConfig.stateSource().getJournalSource(),
                        commonConfig.stateSource().getNoJournalSource(),
config.databasePruneDepth());
            } else {
                pruneManager = new PruneManager(null, null, null, -1); // dummy
            }
```

```
        }
        return pruneManager;
    }
}
```

123:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\config\SystemProperties.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.spongycastle.util.encoders.Hex;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.reflect.Method;
import java.math.BigInteger;
import java.net.InetAddress;
import java.net.Socket;
import java.net.URL;
import java.util.*;
import java.util.function.Function;

import static org.ethereum.util.ByteUtil.toHexString;

/**
 * Utility class to retrieve property values from the ethereumj.conf files
 * <p>
 * The properties are taken from different sources and merged in the following order
 * (the config option from the next source overrides option from previous):
 * - resource ethereumj.conf : normally used as a reference config with default values
 * and shouldn't be changed
 * - system property : each config entry might be altered via -D VM option
 * - [user dir]/config/ethereumj.conf
 * - config specified with the -Dethereumj.conf.file=[file.conf] VM option
 * - CLI options
 *
 * @author Roman Mandeleil
```

```
 * @since 22.05.2014
 */
public class SystemProperties {
    private static Logger logger = LoggerFactory.getLogger("general");

    public final static String PROPERTY_DB_DIR = "database.dir";
    public final static String PROPERTY_LISTEN_PORT = "peer.listen.port";
    public final static String PROPERTY_PEER_ACTIVE = "peer.active";
    public final static String PROPERTY_DB_RESET = "database.reset";
    public final static String PROPERTY_PEER_DISCOVERY_ENABLED =
"peer.discovery.enabled";

    /* Testing */
    private final static Boolean DEFAULT_VMTEST_LOAD_LOCAL = false;
    private final static String DEFAULT_BLOCKS_LOADER = "";

    private static SystemProperties CONFIG;
    private static boolean useOnlySpringConfig = false;
    private String generatedNodePrivateKey;

    /**
     * Returns the static config instance. If the config is passed
     * as a Spring bean by the application this instance shouldn't
     * be used
     * This method is mainly used for testing purposes
     * (Autowired fields are initialized with this static instance
     * but when running within Spring context they replaced with the
     * bean config instance)
     */
    public static SystemProperties getDefault() {
        return useOnlySpringConfig ? null : getSpringDefault();
    }

    static SystemProperties getSpringDefault() {
        if (CONFIG == null) {
            CONFIG = new SystemProperties();
        }
        return CONFIG;
    }

    public static void resetToDefault() {
        CONFIG = null;
```

```
}

/**
 * Used mostly for testing purposes to ensure the application
 * refers only to the config passed as a Spring bean.
 * If this property is set to true {@link #getDefault()} returns null
 */
public static void setUseOnlySpringConfig(boolean useOnlySpringConfig) {
    SystemProperties.useOnlySpringConfig = useOnlySpringConfig;
}

static boolean isUseOnlySpringConfig() {
    return useOnlySpringConfig;
}

/**
 * Marks config accessor methods which need to be called (for value validation)
 * upon config creation or modification
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
private @interface ValidateMe {
}

private Config config;

// mutable options for tests
private String databaseDir = null;
private Boolean databaseReset = null;
private String projectVersion = null;
private String projectVersionModifier = null;
protected Integer databaseVersion = null;

private String genesisInfo = null;

private String bindIp = null;
private String externalIp = null;

private Boolean syncEnabled = null;
private Boolean discoveryEnabled = null;

private BlockchainNetConfig blockchainConfig;
```

```
private Boolean vmTrace;
private Boolean recordInternalTransactionsData;

public SystemProperties() {
    Map<String, Object> values = new HashMap<>();
    values.put("cache.flush.writeCacheSize", 64);
    values.put("cache.flush.blocks", 0);
    values.put("cache.flush.shortSyncFlush", true);
    values.put("cache.stateCacheSize", 384);
    values.put("crypto.providerName", "SC");
    values.put("crypto.hash.alg256", "ETH-KECCAK-256");
    values.put("crypto.hash.alg512", "ETH-KECCAK-512");
    values.put("database.maxOpenFiles", 512);
    values.put("database.prune.enabled", true);
    values.put("database.prune.maxDepth", 192);
    values.put("keyvalue.datasource", "");
    config = ConfigFactory.parseMap(values);
}

/**
 * Loads resources using given ClassLoader assuming, there could be several resources
 * with the same name
 */
public static List<InputStream> loadResources(
        final String name, final ClassLoader classLoader) throws IOException {
    final List<InputStream> list = new ArrayList<InputStream>();
    final Enumeration<URL> systemResources =
            (classLoader == null ? ClassLoader.getSystemClassLoader() : classLoader)
                    .getResources(name);
    while (systemResources.hasMoreElements()) {
        list.add(systemResources.nextElement().openStream());
    }
    return list;
}

public Config getConfig() {
    return config;
}

/**
 * Puts a new config atop of existing stack making the options
 * in the supplied config overriding existing options
```

```
     * Once put this config can't be removed
     *
     * @param overrideOptions - atop config
     */
    public void overrideParams(Config overrideOptions) {
        config = overrideOptions.withFallback(config);
        validateConfig();
    }

    /**
     * Puts a new config atop of existing stack making the options
     * in the supplied config overriding existing options
     * Once put this config can't be removed
     *
     * @param keyValuePairs [name] [value] [name] [value] ...
     */
    public void overrideParams(String... keyValuePairs) {
        if (keyValuePairs.length % 2 != 0) {
            throw new RuntimeException("Odd argument number");
        }
        Map<String, String> map = new HashMap<>();
        for (int i = 0; i < keyValuePairs.length; i += 2) {
            map.put(keyValuePairs[i], keyValuePairs[i + 1]);
        }
        overrideParams(map);
    }

    /**
     * Puts a new config atop of existing stack making the options
     * in the supplied config overriding existing options
     * Once put this config can't be removed
     *
     * @param cliOptions -  command line options to take presidency
     */
    public void overrideParams(Map<String, ?> cliOptions) {
        Config cliConf = ConfigFactory.parseMap(cliOptions);
        overrideParams(cliConf);
    }

    private void validateConfig() {
        for (Method method : getClass().getMethods()) {
            try {
```

```
            if (method.isAnnotationPresent(ValidateMe.class)) {
                method.invoke(this);
            }
        } catch (Exception e) {
            throw new RuntimeException("Error validating config method: " + method, e);
        }
    }
}

/**
 * Builds config from the list of config references in string doing following actions:
 * 1) Splits input by "," to several strings
 * 2) Uses parserFunc to create config from each string reference
 * 3) Merges configs, applying them in the same order as in input, so last overrides first
 *
 * @param input      String with list of config references separated by ",", null or one reference works fine
 * @param parserFunc Function to apply to each reference, produces config from it
 * @return Merged config
 */
protected Config mergeConfigs(String input, Function<String, Config> parserFunc) {
    Config config = ConfigFactory.empty();
    if (input != null && !input.isEmpty()) {
        String[] list = input.split(",");
        for (int i = list.length - 1; i >= 0; --i) {
            config = config.withFallback(parserFunc.apply(list[i]));
        }
    }

    return config;
}

public <T> T getProperty(String propName, T defaultValue) {
    if (!config.hasPath(propName)) {
        return defaultValue;
    }
    String string = config.getString(propName);
    if (string.trim().isEmpty()) {
        return defaultValue;
    }
    return (T) config.getAnyRef(propName);
}
```

```
    public BlockchainNetConfig getBlockchainConfig() {
        if (blockchainConfig == null) {
            blockchainConfig = new BlockchainNetConfig() {
                @Override
                public Constants getCommonConstants() {
                    return new Constants();
                }
            };
        }
        return blockchainConfig;
    }

    public void setBlockchainConfig(BlockchainNetConfig config) {
        blockchainConfig = config;
    }

    @ValidateMe
    public boolean peerDiscovery() {
        return discoveryEnabled == null ? config.getBoolean("peer.discovery.enabled") :
discoveryEnabled;
    }

    public void setDiscoveryEnabled(Boolean discoveryEnabled) {
        this.discoveryEnabled = discoveryEnabled;
    }

    @ValidateMe
    public boolean peerDiscoveryPersist() {
        return config.getBoolean("peer.discovery.persist");
    }

    @ValidateMe
    public int peerDiscoveryWorkers() {
        return config.getInt("peer.discovery.workers");
    }

    @ValidateMe
    public int peerDiscoveryTouchPeriod() {
        return config.getInt("peer.discovery.touchPeriod");
    }
```

```
@ValidateMe
public int peerDiscoveryTouchMaxNodes() {
    return config.getInt("peer.discovery.touchMaxNodes");
}

@ValidateMe
public int peerConnectionTimeout() {
    return config.getInt("peer.connection.timeout") * 1000;
}

@ValidateMe
public int transactionApproveTimeout() {
    return config.getInt("transaction.approve.timeout") * 1000;
}

@ValidateMe
public List<String> peerDiscoveryIPList() {
    return config.getStringList("peer.discovery.ip.list");
}

@ValidateMe
public boolean databaseReset() {
    return databaseReset == null ? config.getBoolean("database.reset") : databaseReset;
}

public void setDatabaseReset(Boolean reset) {
    databaseReset = reset;
}

@ValidateMe
public long databaseResetBlock() {
    return config.getLong("database.resetBlock");
}

@ValidateMe
public boolean databaseFromBackup() {
    return config.getBoolean("database.fromBackup");
}

@ValidateMe
public int databasePruneDepth() {
    return config.getBoolean("database.prune.enabled") ?
```

```
config.getInt("database.prune.maxDepth") : -1;
    }

    @ValidateMe
    public Integer blockQueueSize() {
        return config.getInt("cache.blockQueueSize") * 1024 * 1024;
    }

    @ValidateMe
    public Integer headerQueueSize() {
        return config.getInt("cache.headerQueueSize") * 1024 * 1024;
    }

    @ValidateMe
    public Integer peerChannelReadTimeout() {
        return config.getInt("peer.channel.read.timeout");
    }

    @ValidateMe
    public Integer traceStartBlock() {
        return config.getInt("trace.startblock");
    }

    @ValidateMe
    public boolean recordBlocks() {
        return config.getBoolean("record.blocks");
    }

    @ValidateMe
    public boolean dumpFull() {
        return config.getBoolean("dump.full");
    }

    @ValidateMe
    public String dumpDir() {
        return config.getString("dump.dir");
    }

    @ValidateMe
    public String dumpStyle() {
        return config.getString("dump.style");
    }
```

```
@ValidateMe
public int dumpBlock() {
    return config.getInt("dump.block");
}

@ValidateMe
public String databaseDir() {
    return databaseDir == null ? config.getString("database.dir") : databaseDir;
}

public String ethashDir() {
    return config.hasPath("ethash.dir") ? config.getString("ethash.dir") : databaseDir();
}

public void setDataBaseDir(String dataBaseDir) {
    this.databaseDir = dataBaseDir;
}

@ValidateMe
public boolean dumpCleanOnRestart() {
    return config.getBoolean("dump.clean.on.restart");
}

@ValidateMe
public boolean playVM() {
    return config.getBoolean("play.vm");
}

@ValidateMe
public boolean blockChainOnly() {
    return config.getBoolean("blockchain.only");
}

@ValidateMe
public int syncPeerCount() {
    return config.getInt("sync.peer.count");
}

public Integer syncVersion() {
    if (!config.hasPath("sync.version")) {
        return null;
```

```
    }
    return config.getInt("sync.version");
}

@ValidateMe
public boolean exitOnBlockConflict() {
    return config.getBoolean("sync.exitOnBlockConflict");
}

@ValidateMe
public String projectVersion() {
    return projectVersion;
}

@ValidateMe
public Integer databaseVersion() {
    return databaseVersion;
}

@ValidateMe
public String projectVersionModifier() {
    return projectVersionModifier;
}

@ValidateMe
public String helloPhrase() {
    return config.getString("hello.phrase");
}

@ValidateMe
public String rootHashStart() {
    return config.hasPath("root.hash.start") ? config.getString("root.hash.start") : null;
}

@ValidateMe
public List<String> peerCapabilities() {
    return config.getStringList("peer.capabilities");
}

@ValidateMe
public boolean vmTrace() {
    return vmTrace == null ? (vmTrace = config.getBoolean("vm.structured.trace")) : vmTrace;
```

```
}

@ValidateMe
public boolean vmTraceCompressed() {
    return config.getBoolean("vm.structured.compressed");
}

@ValidateMe
public int vmTraceInitStorageLimit() {
    return config.getInt("vm.structured.initStorageLimit");
}

@ValidateMe
public int cacheFlushBlocks() {
    return config.getInt("cache.flush.blocks");
}

@ValidateMe
public String vmTraceDir() {
    return config.getString("vm.structured.dir");
}

public String customSolcPath() {
    return config.hasPath("solc.path") ? config.getString("solc.path") : null;
}

@ValidateMe
public int networkId() {
    return config.getInt("peer.networkId");
}

@ValidateMe
public int maxActivePeers() {
    return config.getInt("peer.maxActivePeers");
}

@ValidateMe
public boolean eip8() {
    return config.getBoolean("peer.p2p.eip8");
}

@ValidateMe
```

```
    public int listenPort() {
        return config.getInt("peer.listen.port");
    }


    /**
     * This can be a blocking call with long timeout (thus no ValidateMe)
     */
    public String bindIp() {
        if (!config.hasPath("peer.discovery.bind.ip") ||
config.getString("peer.discovery.bind.ip").trim().isEmpty()) {
            if (bindIp == null) {
                logger.debug("Bind address wasn't set, Punching to identify it...");
                try (Socket s = new Socket("www.google.com", 80)) {
                    bindIp = s.getLocalAddress().getHostAddress();
                    logger.debug("UDP local bound to: {}", bindIp);
                } catch (IOException e) {
                    logger.warn("Can't get bind IP. Fall back to 0.0.0.0: " + e);
                    bindIp = "0.0.0.0";
                }
            }
            return bindIp;
        } else {
            return config.getString("peer.discovery.bind.ip").trim();
        }
    }

    /**
     * This can be a blocking call with long timeout (thus no ValidateMe)
     */
    public String externalIp() {
        if (!config.hasPath("peer.discovery.external.ip") ||
config.getString("peer.discovery.external.ip").trim().isEmpty()) {
            if (externalIp == null) {
                logger.debug("External IP wasn't set, using checkip.amazonaws.com to identify it...");
                try {
                    BufferedReader in = new BufferedReader(new InputStreamReader(
                            new URL("http://checkip.amazonaws.com").openStream()));
                    externalIp = in.readLine();
                    if (externalIp == null || externalIp.trim().isEmpty()) {
                        throw new IOException("Invalid address: '" + externalIp + "'");
                    }
```

```
                try {
                    InetAddress.getByName(externalIp);
                } catch (Exception e) {
                    throw new IOException("Invalid address: '" + externalIp + "'");
                }
                logger.debug("External address identified: {}", externalIp);
            } catch (IOException e) {
                externalIp = bindIp();
                logger.warn("Can't get external IP. Fall back to peer.bind.ip: " + externalIp + " :" + e);
            }
        }
        return externalIp;

    } else {
        return config.getString("peer.discovery.external.ip").trim();
    }
}

@ValidateMe
public String getKeyValueDataSource() {
    return config.getString("keyvalue.datasource");
}

@ValidateMe
public boolean isSyncEnabled() {
    return this.syncEnabled == null ? config.getBoolean("sync.enabled") : syncEnabled;
}

public void setSyncEnabled(Boolean syncEnabled) {
    this.syncEnabled = syncEnabled;
}

@ValidateMe
public boolean isFastSyncEnabled() {
    return isSyncEnabled() && config.getBoolean("sync.fast.enabled");
}

@ValidateMe
public byte[] getFastSyncPivotBlockHash() {
    if (!config.hasPath("sync.fast.pivotBlockHash")) {
        return null;
    }
```

```
    byte[] ret = Hex.decode(config.getString("sync.fast.pivotBlockHash"));
    if (ret.length != 32) {
        throw new RuntimeException("Invalid block hash length: " + toHexString(ret));
    }
    return ret;
}

@ValidateMe
public boolean fastSyncBackupState() {
    return config.getBoolean("sync.fast.backupState");
}

@ValidateMe
public boolean fastSyncSkipHistory() {
    return config.getBoolean("sync.fast.skipHistory");
}

@ValidateMe
public int makeDoneByTimeout() {
    return config.getInt("sync.makeDoneByTimeout");
}



@ValidateMe
public boolean isPublicHomeNode() {
    return config.getBoolean("peer.discovery.public.home.node");
}

@ValidateMe
public String genesisInfo() {
    return genesisInfo == null ? config.getString("genesis") : genesisInfo;
}

@ValidateMe
public int txOutdatedThreshold() {
    return config.getInt("transaction.outdated.threshold");
}

public void setGenesisInfo(String genesisInfo) {
    this.genesisInfo = genesisInfo;
}
```

```
@ValidateMe
public boolean minerStart() {
    return config.getBoolean("mine.start");
}

@ValidateMe
public byte[] getMinerCoinbase() {
    String sc = config.getString("mine.coinbase");
    byte[] c = ByteUtil.hexStringToBytes(sc);
    if (c.length != 20) {
        throw new RuntimeException("mine.coinbase has invalid value: '" + sc + "'");
    }
    return c;
}

@ValidateMe
public byte[] getMineExtraData() {
    byte[] bytes;
    if (config.hasPath("mine.extraDataHex")) {
        bytes = Hex.decode(config.getString("mine.extraDataHex"));
    } else {
        bytes = config.getString("mine.extraData").getBytes();
    }
    if (bytes.length > 32) {
        throw new RuntimeException("mine.extraData exceed 32 bytes length: " + bytes.length);
    }
    return bytes;
}

@ValidateMe
public BigInteger getMineMinGasPrice() {
    return new BigInteger(config.getString("mine.minGasPrice"));
}

@ValidateMe
public long getMineMinBlockTimeoutMsec() {
    return config.getLong("mine.minBlockTimeoutMsec");
}

@ValidateMe
public int getMineCpuThreads() {
    return config.getInt("mine.cpuMineThreads");
```

```
}

@ValidateMe
public boolean isMineFullDataset() {
    return config.getBoolean("mine.fullDataSet");
}

@ValidateMe
public String getCryptoProviderName() {
    return config.getString("crypto.providerName");
}

@ValidateMe
public boolean recordInternalTransactionsData() {
    if (recordInternalTransactionsData == null) {
        recordInternalTransactionsData = config.getBoolean("record.internal.transactions.data");
    }
    return recordInternalTransactionsData;
}

public void setRecordInternalTransactionsData(Boolean recordInternalTransactionsData) {
    this.recordInternalTransactionsData = recordInternalTransactionsData;
}

@ValidateMe
public String getHash256AlgName() {
    return config.getString("crypto.hash.alg256");
}

@ValidateMe
public String getHash512AlgName() {
    return config.getString("crypto.hash.alg512");
}

@ValidateMe
public String getEthashMode() {
    return config.getString("sync.ethash");
}


public String dump() {
    return config.root().render(ConfigRenderOptions.defaults().setComments(false));
```

```
    }

    /*
     *
     * Testing
     *
     */
    public boolean vmTestLoadLocal() {
        return config.hasPath("GitHubTests.VMTest.loadLocal") ?
                config.getBoolean("GitHubTests.VMTest.loadLocal") :
DEFAULT_VMTEST_LOAD_LOCAL;
    }

    public String blocksLoader() {
        return config.hasPath("blocks.loader") ?
                config.getString("blocks.loader") : DEFAULT_BLOCKS_LOADER;
    }

    public String githubTestsPath() {
        return config.hasPath("GitHubTests.testPath") ?
                config.getString("GitHubTests.testPath") : "";
    }

    public boolean githubTestsLoadLocal() {
        return config.hasPath("GitHubTests.testPath") &&
                !config.getString("GitHubTests.testPath").isEmpty();
    }

}
```

124:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\core\AccountState.java

```
import java.math.BigInteger;

import static org.ethereum.crypto.HashUtil.EMPTY_DATA_HASH;
import static org.ethereum.crypto.HashUtil.EMPTY_TRIE_HASH;
import static org.ethereum.util.ByteUtil.toHexString;
import static org.ethereum.util.FastByteComparisons.equal;

public class AccountState {
```

```
    private byte[] rlpEncoded;

    /* A value equal to the number of transactions sent
     * from this address, or, in the case of contract accounts,
     * the number of contract-creations made by this account */
    private final BigInteger nonce;

    /* A scalar value equal to the number of Wei owned by this address */
    private final BigInteger balance;

    /* A 256-bit hash of the root node of a trie structure
     * that encodes the storage contents of the contract,
     * itself a simple mapping between byte arrays of size 32.
     * The hash is formally denoted [a] s .
     *
     * Since I typically wish to refer not to the trie's root hash
     * but to the underlying set of key/value pairs stored within,
     * I define a convenient equivalence TRIE ([a] s )  [a] s .
     * It shall be understood that [a] s is not a 'physical' member
     * of the account and does not contribute to its later serialisation */
    private final byte[] stateRoot;

    /* The hash of the EVM code of this contract—this is the code
     * that gets executed should this address receive a message call;
     * it is immutable and thus, unlike all other fields, cannot be changed
     * after construction. All such code fragments are contained in
     * the state database under their corresponding hashes for later
     * retrieval */
    private final byte[] codeHash;

    private final byte[] owner;

    public AccountState(BigInteger nonce, BigInteger balance) {
        this(nonce, balance, EMPTY_TRIE_HASH, EMPTY_DATA_HASH, null);
    }

    public AccountState(BigInteger nonce, BigInteger balance, byte[] owner) {
        this(nonce, balance, EMPTY_TRIE_HASH, EMPTY_DATA_HASH, owner);
    }

    public AccountState(BigInteger nonce, BigInteger balance, byte[] stateRoot, byte[] codeHash,
byte[] owner) {
```

```
        this.nonce = nonce;
        this.balance = balance;
        this.stateRoot = stateRoot == EMPTY_TRIE_HASH || equal(stateRoot,
EMPTY_TRIE_HASH) ? EMPTY_TRIE_HASH : stateRoot;
        this.codeHash = codeHash == EMPTY_DATA_HASH || equal(codeHash,
EMPTY_DATA_HASH) ? EMPTY_DATA_HASH : codeHash;
        this.owner = owner == EMPTY_DATA_HASH || equal(owner, EMPTY_DATA_HASH) ?
EMPTY_DATA_HASH : owner;
    }

    public AccountState(byte[] rlpData) {
        this.rlpEncoded = rlpData;

        RLPList items = (RLPList) RLP.decode2(rlpEncoded).get(0);
        this.nonce = ByteUtil.bytesToBigInteger(items.get(0).getRLPData());
        this.balance = ByteUtil.bytesToBigInteger(items.get(1).getRLPData());
        this.stateRoot = items.get(2).getRLPData();
        this.codeHash = items.get(3).getRLPData();
        this.owner = items.get(4).getRLPData();
    }

    public BigInteger getNonce() {
        return nonce;
    }

    public AccountState withNonce(BigInteger nonce) {
        return new AccountState(nonce, balance, stateRoot, codeHash, owner);
    }

    public byte[] getStateRoot() {
        return stateRoot;
    }

    public AccountState withStateRoot(byte[] stateRoot) {
        return new AccountState(nonce, balance, stateRoot, codeHash, owner);
    }

    public AccountState withIncrementedNonce() {
        return new AccountState(nonce.add(BigInteger.ONE), balance, stateRoot, codeHash,
owner);
    }
```

```
public byte[] getCodeHash() {
    return codeHash;
}

public AccountState withCodeHash(byte[] codeHash) {
    return new AccountState(nonce, balance, stateRoot, codeHash, owner);
}

public BigInteger getBalance() {
    return balance;
}

public AccountState withBalanceIncrement(BigInteger value) {
    return new AccountState(nonce, balance.add(value), stateRoot, codeHash, owner);
}

public byte[] getOwner() {
    return owner;
}

public AccountState withOwner(byte[] owner) {
    return new AccountState(nonce, balance, stateRoot, codeHash, owner);
}

public byte[] getEncoded() {
    if (rlpEncoded == null) {
        byte[] nonce = RLP.encodeBigInteger(this.nonce);
        byte[] balance = RLP.encodeBigInteger(this.balance);
        byte[] stateRoot = RLP.encodeElement(this.stateRoot);
        byte[] codeHash = RLP.encodeElement(this.codeHash);
        byte[] owner = RLP.encodeElement(this.owner);
        this.rlpEncoded = RLP.encodeList(nonce, balance, stateRoot, codeHash, owner);
    }
    return rlpEncoded;
}

public boolean isEmpty() {
    return FastByteComparisons.equal(codeHash, EMPTY_DATA_HASH) &&
            BigInteger.ZERO.equals(balance) &&
            BigInteger.ZERO.equals(nonce);
}
```

```
    @Override
    public String toString() {
        String ret = "  Nonce: " + this.getNonce().toString() + "\n" +
                "  Balance: " + getBalance() + "\n" +
                "  State Root: " + toHexString(this.getStateRoot()) + "\n" +
                "  Code Hash: " + toHexString(this.getCodeHash()) + "\n" +
                "  Owner: " + toHexString(this.getOwner());
        return ret;
    }
}
```

125:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\core\Block.java

```
import org.spongycastle.util.Arrays;

import java.util.ArrayList;
import java.util.List;

import static org.ethereum.util.ByteUtil.toHexString;

/**
 * The block in Ethereum is the collection of relevant pieces of information
 * (known as the blockheader), H, together with information corresponding to
 * the comprised transactions, R, and a set of other blockheaders U that are known
 * to have a parent equal to the present block’s parent’s parent
 * (such blocks are known as uncles).
 *
 * @author Roman Mandeleil
 * @author Nick Savers
 * @since 20.05.2014
 */
public class Block {

    private static final Logger logger = LoggerFactory.getLogger("blockchain");

    private BlockHeader header;

    /* Private */

    private byte[] rlpEncoded;
    private boolean parsed = false;
```

```
/* Constructors */

private Block() {
}

public Block(byte[] rawData) {
    logger.debug("new from [" + toHexString(rawData) + "]");
    this.rlpEncoded = rawData;
}

public Block(byte[] parentHash, byte[] hash, long number) {
    this.header = new BlockHeader(parentHash, hash, number);
    this.parsed = true;
}

private synchronized void parseRLP() {
    if (parsed) {
        return;
    }

    RLPList params = RLP.decode2(rlpEncoded);
    RLPList block = (RLPList) params.get(0);

    // Parse Header
    RLPList header = (RLPList) block.get(0);
    this.header = new BlockHeader(header);
    this.parsed = true;
}

public BlockHeader getHeader() {
    parseRLP();
    return this.header;
}

public byte[] getHash() {
    parseRLP();
    return this.header.getHash();
}

public byte[] getParentHash() {
    parseRLP();
```

```
        return this.header.getParentHash();
    }

    public long getNumber() {
        parseRLP();
        return this.header.getNumber();
    }

    public boolean isEqual(Block block) {
        return Arrays.areEqual(this.getHash(), block.getHash());
    }

    public byte[] getEncoded() {
        if (rlpEncoded == null) {
            byte[] header = this.header.getEncoded();

            List<byte[]> block = getBodyElements();
            block.add(0, header);
            byte[][] elements = block.toArray(new byte[block.size()][]);

            this.rlpEncoded = RLP.encodeList(elements);
        }
        return rlpEncoded;
    }

    private List<byte[]> getBodyElements() {
        parseRLP();

        List<byte[]> body = new ArrayList<>();

        return body;
    }

}

126:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\org\ethereum\core\BlockHeader.java
import org.spongycastle.util.Arrays;

import java.math.BigInteger;

import static org.ethereum.util.ByteUtil.toHexString;
```

```
/**
 * Block header is a value object containing
 * the basic information of a block
 */
public class BlockHeader {

    private byte[] parentHash;
    private byte[] hash;
    private long number;

    public BlockHeader(byte[] encoded) {
        this((RLPList) RLP.decode2(encoded).get(0));
    }

    public BlockHeader(RLPList rlpHeader) {
        this.parentHash = rlpHeader.get(0).getRLPData();
        this.hash = rlpHeader.get(1).getRLPData();
        byte[] nrBytes = rlpHeader.get(2).getRLPData();
        this.number = ByteUtil.byteArrayToLong(nrBytes);
    }

    public BlockHeader(byte[] parentHash, byte[] hash, long number) {
        this.parentHash = parentHash;
        this.hash = hash;
        this.number = number;
    }

    public byte[] getParentHash() {
        return parentHash;
    }

    public long getNumber() {
        return number;
    }

    public byte[] getHash() {
        return hash;
    }

    public byte[] getEncoded() {
        return this.getEncoded(true); // with nonce
```

```
    }

    public byte[] getEncoded(boolean withNonce) {
        byte[] parentHash = RLP.encodeElement(this.parentHash);
        byte[] hash = RLP.encodeElement(this.hash);
        byte[] number = RLP.encodeBigInteger(BigInteger.valueOf(this.number));
        return RLP.encodeList(parentHash, hash, number);
    }

    @Override
    public String toString() {
        return toStringWithSuffix("\n");
    }

    private String toStringWithSuffix(final String suffix) {
        StringBuilder toStringBuff = new StringBuilder();
        toStringBuff.append("  hash=").append(toHexString(hash)).append(suffix);
        toStringBuff.append("  parentHash=").append(toHexString(parentHash)).append(suffix);
        toStringBuff.append("  number=").append(number).append(suffix);
        return toStringBuff.toString();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        BlockHeader that = (BlockHeader) o;
        return FastByteComparisons.equal(getHash(), that.getHash());
    }

    @Override
    public int hashCode() {
        return Arrays.hashCode(getHash());
    }
}
```

127:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\core\Repository.java

```
import java.math.BigInteger;
import java.util.HashMap;
import java.util.Set;

/**
 * @author Roman Mandeleil
 * @since 08.09.2014
 */
public interface Repository extends org.ethereum.facade.Repository {

    /**
     * Create a new account in the database
     *
     * @param addr of the contract
     * @return newly created account state
     */
    AccountState createAccount(byte[] addr, byte[] creater);


    /**
     * @param addr - account to check
     * @return - true if account exist,
     * false otherwise
     */
    @Override
    boolean isExist(byte[] addr);

    /**
     * Retrieve an account
     *
     * @param addr of the account
     * @return account state as stored in the database
     */
    AccountState getAccountState(byte[] addr);

    /**
     * Deletes the account
     *
     * @param addr of the account
     */
    void delete(byte[] addr);
```

```
/**
 * Increase the account nonce of the given account by one
 *
 * @param addr of the account
 * @return new value of the nonce
 */
BigInteger increaseNonce(byte[] addr);

/**
 * Sets the account nonce of the given account
 *
 * @param addr  of the account
 * @param nonce new nonce
 * @return new value of the nonce
 */
BigInteger setNonce(byte[] addr, BigInteger nonce);

/**
 * Get current nonce of a given account
 *
 * @param addr of the account
 * @return value of the nonce
 */
@Override
BigInteger getNonce(byte[] addr);

/**
 * Retrieve contract details for a given account from the database
 *
 * @param addr of the account
 * @return new contract details
 */
ContractDetails getContractDetails(byte[] addr);

boolean hasContractDetails(byte[] addr);

/**
 * Store code associated with an account
 *
 * @param addr for the account
 * @param code that will be associated with this account
 */
```

```
void saveCode(byte[] addr, byte[] code);

/**
 * Retrieve the code associated with an account
 *
 * @param addr of the account
 * @return code in byte-array format
 */
@Override
byte[] getCode(byte[] addr);

/**
 * Retrieve the code hash associated with an account
 *
 * @param addr of the account
 * @return code hash
 */
byte[] getCodeHash(byte[] addr);

/**
 * Put a value in storage of an account at a given key
 *
 * @param addr  of the account
 * @param key   of the data to store
 * @param value is the data to store
 */
void addStorageRow(byte[] addr, DataWord key, DataWord value);


/**
 * Retrieve storage value from an account for a given key
 *
 * @param addr of the account
 * @param key  associated with this value
 * @return data in the form of a <code>DataWord</code>
 */
@Override
DataWord getStorageValue(byte[] addr, DataWord key);


/**
 * Retrieve balance of an account
```

```
     *
     * @param addr of the account
     * @return balance of the account as a <code>BigInteger</code> value
     */
    @Override
    BigInteger getBalance(byte[] addr);

    /**
     * Add value to the balance of an account
     *
     * @param addr  of the account
     * @param value to be added
     * @return new balance of the account
     */
    BigInteger addBalance(byte[] addr, BigInteger value);

    /**
     * @return Returns set of all the account addresses
     */
    Set<byte[]> getAccountsKeys();


    /**
     * Dump the full state of the current repository into a file with JSON format
     * It contains all the contracts/account, their attributes and
     *
     * @param block    of the current state
     * @param gasUsed  the amount of gas used in the block until that point
     * @param txNumber is the number of the transaction for which the dump has to be made
     * @param txHash   is the hash of the given transaction.
     *                 If null, the block state post coinbase reward is dumped.
     */
    void dumpState(Block block, long gasUsed, int txNumber, byte[] txHash);

    /**
     * Save a snapshot and start tracking future changes
     *
     * @return the tracker repository
     */
    Repository startTracking();

    void flush();
```

```
void flushNoReconnect();


/**
 * Store all the temporary changes made
 * to the repository in the actual database
 */
void commit();

/**
 * Undo all the changes made so far
 * to a snapshot of the repository
 */
void rollback();

/**
 * Return to one of the previous snapshots
 * by moving the root.
 *
 * @param root - new root
 */
void syncToRoot(byte[] root);

/**
 * Check to see if the current repository has an open connection to the database
 *
 * @return <tt>true</tt> if connection to database is open
 */
boolean isClosed();

/**
 * Close the database
 */
void close();

/**
 * Reset
 */
void reset();

void updateBatch(HashMap<ByteArrayWrapper, AccountState> accountStates,
```

```
                HashMap<ByteArrayWrapper, ContractDetails> contractDetailes);


    byte[] getRoot();

    void loadAccount(byte[] addr, HashMap<ByteArrayWrapper, AccountState> cacheAccounts,
                HashMap<ByteArrayWrapper, ContractDetails> cacheDetails);

    Repository getSnapshotTo(byte[] root);

    /**
     * Clones repository so changes made to this repository are
     * not reflected in its clone.
     */
    Repository clone();
}

128:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\org\ethereum\crypto\cryptohash\Digest.java
 * interface somewhat mimics the standard {@code
 * java.security.MessageDigest} class. We do not extend that class in
 * order to provide compatibility with reduced Java implementations such
 * as J2ME. Implementing a {@code java.security.Provider} compatible
 * with Sun's JCA ought to be easy.</p>
 *
 * <p>A {@code Digest} object maintains a running state for a hash
 * function computation. Data is inserted with {@code update()} calls;
 * the result is obtained from a {@code digest()} method (where some
 * final data can be inserted as well). When a digest output has been
 * produced, the objet is automatically resetted, and can be used
 * immediately for another digest operation. The state of a computation
 * can be cloned with the {@link #copy} method; this can be used to get
 * a partial hash result without interrupting the complete
 * computation.</p>
 *
 * <p>{@code Digest} objects are stateful and hence not thread-safe;
 * however, distinct {@code Digest} objects can be accessed concurrently
 * without any problem.</p>
 *
 * <pre>
 * ==========================(LICENSE BEGIN)============================
 *
```

```
* Copyright (c) 2007-2010  Projet RNRT SAPHIR
*
* Permission is hereby granted, free of charge, to any person obtaining
* a copy of this software and associated documentation files (the
* "Software"), to deal in the Software without restriction, including
* without limitation the rights to use, copy, modify, merge, publish,
* distribute, sublicense, and/or sell copies of the Software, and to
* permit persons to whom the Software is furnished to do so, subject to
* the following conditions:
*
* The above copyright notice and this permission notice shall be
* included in all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
* IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
* CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
* TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
* ============================(LICENSE END)=============================
* </pre>
*
* @author Thomas Pornin &lt;thomas.pornin@cryptolog.com&gt;
* @version $Revision: 232 $
*/

public interface Digest {

    /**
     * Insert one more input data byte.
     *
     * @param in the input byte
     */
    void update(byte in);

    /**
     * Insert some more bytes.
     *
     * @param inbuf the data bytes
     */
```

```
void update(byte[] inbuf);

/**
 * Insert some more bytes.
 *
 * @param inbuf the data buffer
 * @param off   the data offset in {@code inbuf}
 * @param len   the data length (in bytes)
 */
void update(byte[] inbuf, int off, int len);

/**
 * Finalize the current hash computation and return the hash value
 * in a newly-allocated array. The object is resetted.
 *
 * @return the hash output
 */
byte[] digest();

/**
 * Input some bytes, then finalize the current hash computation
 * and return the hash value in a newly-allocated array. The object
 * is resetted.
 *
 * @param inbuf the input data
 * @return the hash output
 */
byte[] digest(byte[] inbuf);

/**
 * Finalize the current hash computation and store the hash value
 * in the provided output buffer. The {@code len} parameter
 * contains the maximum number of bytes that should be written;
 * no more bytes than the natural hash function output length will
 * be produced. If {@code len} is smaller than the natural
 * hash output length, the hash output is truncated to its first
 * {@code len} bytes. The object is resetted.
 *
 * @param outbuf the output buffer
 * @param off    the output offset within {@code outbuf}
 * @param len    the requested hash output length (in bytes)
 * @return the number of bytes actually written in {@code outbuf}
```

```
 */
int digest(byte[] outbuf, int off, int len);

/**
 * Get the natural hash function output length (in bytes).
 *
 * @return the digest output length (in bytes)
 */
int getDigestLength();

/**
 * Reset the object: this makes it suitable for a new hash
 * computation. The current computation, if any, is discarded.
 */
void reset();

/**
 * Clone the current state. The returned object evolves independantly
 * of this object.
 *
 * @return the clone
 */
Digest copy();

/**
 * <p>Return the "block length" for the hash function. This
 * value is naturally defined for iterated hash functions
 * (Merkle-Damgard). It is used in HMAC (that's what the
 * <a href="http://tools.ietf.org/html/rfc2104">HMAC specification</a>
 * names the "{@code B}" parameter).</p>
 *
 * <p>If the function is "block-less" then this function may
 * return {@code -n} where {@code n} is an integer such that the
 * block length for HMAC ("{@code B}") will be inferred from the
 * key length, by selecting the smallest multiple of {@code n}
 * which is no smaller than the key length. For instance, for
 * the Fugue-xxx hash functions, this function returns -4: the
 * virtual block length B is the HMAC key length, rounded up to
 * the next multiple of 4.</p>
 *
 * @return the internal block length (in bytes), or {@code -n}
 */
```

```
    int getBlockLength();

    /**
     * <p>Get the display name for this function (e.g. {@code "SHA-1"}
     * for SHA-1).</p>
     *
     * @see Object
     */
    @Override
    String toString();
}
```

129:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\cryptohash\DigestEngine.java

```
/**
 * <p>This class is a template which can be used to implement hash
 * functions. It takes care of some of the API, and also provides an
 * internal data buffer whose length is equal to the hash function
 * internal block length.</p>
 *
 * <p>Classes which use this template MUST provide a working {@link
 * #getBlockLength} method even before initialization (alternatively,
 * they may define a custom {@link #getInternalBlockLength} which does
 * not call {@link #getBlockLength}. The {@link #getDigestLength} should
 * also be operational from the beginning, but it is acceptable that it
 * returns 0 while the {@link #doInit} method has not been called
 * yet.</p>
 *
 * <pre>
 * ==========================(LICENSE BEGIN)============================
 *
 * Copyright (c) 2007-2010  Projet RNRT SAPHIR
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be
```

```
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 * ============================(LICENSE END)=============================
 * </pre>
 *
 * @author Thomas Pornin &lt;thomas.pornin@cryptolog.com&gt;
 * @version $Revision: 229 $
 */

public abstract class DigestEngine extends MessageDigest implements Digest {

    /**
     * Reset the hash algorithm state.
     */
    @Override
    protected abstract void engineReset();

    /**
     * Process one block of data.
     *
     * @param data the data block
     */
    protected abstract void processBlock(byte[] data);

    /**
     * Perform the final padding and store the result in the
     * provided buffer. This method shall call {@link #flush}
     * and then {@link #update} with the appropriate padding
     * data in order to get the full input data.
     *
     * @param buf the output buffer
     * @param off the output offset
     */
    protected abstract void doPadding(byte[] buf, int off);
```

```
/**
 * This function is called at object creation time; the
 * implementation should use it to perform initialization tasks.
 * After this method is called, the implementation should be ready
 * to process data or meaningfully honour calls such as
 * {@link #engineGetDigestLength}
 */
protected abstract void doInit();

private int digestLen, blockLen, inputLen;
private byte[] inputBuf, outputBuf;
private long blockCount;

/**
 * Instantiate the engine.
 */
public DigestEngine(String alg) {
    super(alg);
    doInit();
    digestLen = engineGetDigestLength();
    blockLen = getInternalBlockLength();
    inputBuf = new byte[blockLen];
    outputBuf = new byte[digestLen];
    inputLen = 0;
    blockCount = 0;
}

private void adjustDigestLen() {
    if (digestLen == 0) {
        digestLen = engineGetDigestLength();
        outputBuf = new byte[digestLen];
    }
}

/**
 * @see org.ethereum.crypto.cryptohash.Digest
 */
@Override
public byte[] digest() {
    adjustDigestLen();
    byte[] result = new byte[digestLen];
```

```
    digest(result, 0, digestLen);
    return result;
}

/**
 * @see org.ethereum.crypto.cryptohash.Digest
 */
@Override
public byte[] digest(byte[] input) {
    update(input, 0, input.length);
    return digest();
}

/**
 * @see org.ethereum.crypto.cryptohash.Digest
 */
@Override
public int digest(byte[] buf, int offset, int len) {
    adjustDigestLen();
    if (len >= digestLen) {
        doPadding(buf, offset);
        reset();
        return digestLen;
    } else {
        doPadding(outputBuf, 0);
        System.arraycopy(outputBuf, 0, buf, offset, len);
        reset();
        return len;
    }
}

/**
 * @see org.ethereum.crypto.cryptohash.Digest
 */
@Override
public void reset() {
    engineReset();
    inputLen = 0;
    blockCount = 0;
}

/**
```

```
     * @see org.ethereum.crypto.cryptohash.Digest
     */
    @Override
    public void update(byte input) {
        inputBuf[inputLen++] = (byte) input;
        if (inputLen == blockLen) {
            processBlock(inputBuf);
            blockCount++;
            inputLen = 0;
        }
    }

    /**
     * @see org.ethereum.crypto.cryptohash.Digest
     */
    @Override
    public void update(byte[] input) {
        update(input, 0, input.length);
    }

    /**
     * @see org.ethereum.crypto.cryptohash.Digest
     */
    @Override
    public void update(byte[] input, int offset, int len) {
        while (len > 0) {
            int copyLen = blockLen - inputLen;
            if (copyLen > len) {
                copyLen = len;
            }
            System.arraycopy(input, offset, inputBuf, inputLen,
                    copyLen);
            offset += copyLen;
            inputLen += copyLen;
            len -= copyLen;
            if (inputLen == blockLen) {
                processBlock(inputBuf);
                blockCount++;
                inputLen = 0;
            }
        }
    }
```

```
/**
 * Get the internal block length. This is the length (in
 * bytes) of the array which will be passed as parameter to
 * {@link #processBlock}. The default implementation of this
 * method calls {@link #getBlockLength} and returns the same
 * value. Overriding this method is useful when the advertised
 * block length (which is used, for instance, by HMAC) is
 * suboptimal with regards to internal buffering needs.
 *
 * @return the internal block length (in bytes)
 */
protected int getInternalBlockLength() {
    return getBlockLength();
}

/**
 * Flush internal buffers, so that less than a block of data
 * may at most be upheld.
 *
 * @return the number of bytes still unprocessed after the flush
 */
protected final int flush() {
    return inputLen;
}

/**
 * Get a reference to an internal buffer with the same size
 * than a block. The contents of that buffer are defined only
 * immediately after a call to {@link #flush()}: if
 * {@link #flush()} return the value {@code n}, then the
 * first {@code n} bytes of the array returned by this method
 * are the {@code n} bytes of input data which are still
 * unprocessed. The values of the remaining bytes are
 * undefined and may be altered at will.
 *
 * @return a block-sized internal buffer
 */
protected final byte[] getBlockBuffer() {
    return inputBuf;
}
```

```
    /**
     * Get the "block count": this is the number of times the
     * {@link #processBlock} method has been invoked for the
     * current hash operation. That counter is incremented
     * <em>after</em> the call to {@link #processBlock}.
     *
     * @return the block count
     */
    protected long getBlockCount() {
        return blockCount;
    }

    /**
     * This function copies the internal buffering state to some
     * other instance of a class extending {@code DigestEngine}.
     * It returns a reference to the copy. This method is intended
     * to be called by the implementation of the {@link #copy}
     * method.
     *
     * @param dest the copy
     * @return the value {@code dest}
     */
    protected Digest copyState(DigestEngine dest) {
        dest.inputLen = inputLen;
        dest.blockCount = blockCount;
        System.arraycopy(inputBuf, 0, dest.inputBuf, 0,
                inputBuf.length);
        adjustDigestLen();
        dest.adjustDigestLen();
        System.arraycopy(outputBuf, 0, dest.outputBuf, 0,
                outputBuf.length);
        return dest;
    }
}
```

130:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\cryptohash\Keccak256.java

```
 * {@link org.ethereum.crypto.cryptohash.Digest} API.</p>
 *
 * <pre>
 * ========================(LICENSE BEGIN)============================
 *
```

```
* Copyright (c) 2007-2010  Projet RNRT SAPHIR
*
* Permission is hereby granted, free of charge, to any person obtaining
* a copy of this software and associated documentation files (the
* "Software"), to deal in the Software without restriction, including
* without limitation the rights to use, copy, modify, merge, publish,
* distribute, sublicense, and/or sell copies of the Software, and to
* permit persons to whom the Software is furnished to do so, subject to
* the following conditions:
*
* The above copyright notice and this permission notice shall be
* included in all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
* IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
* CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
* TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
* ============================(LICENSE END)=============================
* </pre>
*
* @author Thomas Pornin &lt;thomas.pornin@cryptolog.com&gt;
* @version $Revision: 189 $
*/

public class Keccak256 extends KeccakCore {

    /**
     * Create the engine.
     */
    public Keccak256() {
        super("eth-keccak-256");
    }

    /**
     * @see org.ethereum.crypto.cryptohash.Digest
     */
    @Override
    public Digest copy() {
```

```
        return copyState(new Keccak256());
    }

    /**
     * @see org.ethereum.crypto.cryptohash.Digest
     */
    @Override
    public int engineGetDigestLength() {
        return 32;
    }

    @Override
    protected byte[] engineDigest() {
        return null;
    }

    @Override
    protected void engineUpdate(byte arg0) {
    }

    @Override
    protected void engineUpdate(byte[] arg0, int arg1, int arg2) {
    }
}
```

131:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\cryptohash\Keccak512.java

```
 * {@link Digest} API.</p>
 *
 * <pre>
 * ==========================(LICENSE BEGIN)============================
 *
 * Copyright (c) 2007-2010  Projet RNRT SAPHIR
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
```

```
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 * ============================(LICENSE END)=============================
 * </pre>
 *
 * @author Thomas Pornin &lt;thomas.pornin@cryptolog.com&gt;
 * @version $Revision: 189 $
 */

public class Keccak512 extends KeccakCore {

    /**
     * Create the engine.
     */
    public Keccak512() {
        super("eth-keccak-512");
    }

    /**
     * @see Digest
     */
    @Override
    public Digest copy() {
        return copyState(new Keccak512());
    }

    /**
     * @see Digest
     */
    @Override
    public int engineGetDigestLength() {
        return 64;
    }
```

```
    @Override
    protected byte[] engineDigest() {
        return null;
    }

    @Override
    protected void engineUpdate(byte input) {
    }

    @Override
    protected void engineUpdate(byte[] input, int offset, int len) {
    }
}
```

132:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\cryptohash\KeccakCore.java

```
 * algorithm.
 *
 * <pre>
 * ==========================(LICENSE BEGIN)============================
 *
 * Copyright (c) 2007-2010  Projet RNRT SAPHIR
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
 *
 * ============================(LICENSE END)============================
 * </pre>
 *
 * @author Thomas Pornin &lt;thomas.pornin@cryptolog.com&gt;
 * @version $Revision: 258 $
 */

abstract class KeccakCore extends DigestEngine {

    KeccakCore(String alg) {
        super(alg);
    }

    private long[] A;
    private byte[] tmpOut;

    private static final long[] RC = {
            0x0000000000000001L, 0x0000000000008082L,
            0x800000000000808AL, 0x8000000080008000L,
            0x000000000000808BL, 0x0000000080000001L,
            0x8000000080008081L, 0x8000000000008009L,
            0x000000000000008AL, 0x0000000000000088L,
            0x0000000080008009L, 0x000000008000000AL,
            0x000000008000808BL, 0x800000000000008BL,
            0x8000000000008089L, 0x8000000000008003L,
            0x8000000000008002L, 0x8000000000000080L,
            0x000000000000800AL, 0x800000008000000AL,
            0x8000000080008081L, 0x8000000000008080L,
            0x0000000080000001L, 0x8000000080008008L
    };

    /**
     * Encode the 64-bit word {@code val} into the array
     * {@code buf} at offset {@code off}, in little-endian
     * convention (least significant byte first).
     *
     * @param val the value to encode
     * @param buf the destination buffer
     * @param off the destination offset
     */
    private static void encodeLELong(long val, byte[] buf, int off) {
```

```
    buf[off + 0] = (byte) val;
    buf[off + 1] = (byte) (val >>> 8);
    buf[off + 2] = (byte) (val >>> 16);
    buf[off + 3] = (byte) (val >>> 24);
    buf[off + 4] = (byte) (val >>> 32);
    buf[off + 5] = (byte) (val >>> 40);
    buf[off + 6] = (byte) (val >>> 48);
    buf[off + 7] = (byte) (val >>> 56);
}

/**
 * Decode a 64-bit little-endian word from the array {@code buf}
 * at offset {@code off}.
 *
 * @param buf the source buffer
 * @param off the source offset
 * @return the decoded value
 */
private static long decodeLELong(byte[] buf, int off) {
    return (buf[off + 0] & 0xFFL)
            | ((buf[off + 1] & 0xFFL) << 8)
            | ((buf[off + 2] & 0xFFL) << 16)
            | ((buf[off + 3] & 0xFFL) << 24)
            | ((buf[off + 4] & 0xFFL) << 32)
            | ((buf[off + 5] & 0xFFL) << 40)
            | ((buf[off + 6] & 0xFFL) << 48)
            | ((buf[off + 7] & 0xFFL) << 56);
}

/**
 * @see org.ethereum.crypto.cryptohash.DigestEngine
 */
@Override
protected void engineReset() {
    doReset();
}

/**
 * @see org.ethereum.crypto.cryptohash.DigestEngine
 */
@Override
protected void processBlock(byte[] data) {
```

```
/* Input block */
for (int i = 0; i < data.length; i += 8) {
    A[i >>> 3] ^= decodeLELong(data, i);
}

long t0, t1, t2, t3, t4;
long tt0, tt1, tt2, tt3, tt4;
long t, kt;
long c0, c1, c2, c3, c4, bnn;

/*
 * Unrolling four rounds kills performance big time
 * on Intel x86 Core2, in both 32-bit and 64-bit modes
 * (less than 1 MB/s instead of 55 MB/s on x86-64).
 * Unrolling two rounds appears to be fine.
 */
for (int j = 0; j < 24; j += 2) {

    tt0 = A[1] ^ A[6];
    tt1 = A[11] ^ A[16];
    tt0 ^= A[21] ^ tt1;
    tt0 = (tt0 << 1) | (tt0 >>> 63);
    tt2 = A[4] ^ A[9];
    tt3 = A[14] ^ A[19];
    tt0 ^= A[24];
    tt2 ^= tt3;
    t0 = tt0 ^ tt2;

    tt0 = A[2] ^ A[7];
    tt1 = A[12] ^ A[17];
    tt0 ^= A[22] ^ tt1;
    tt0 = (tt0 << 1) | (tt0 >>> 63);
    tt2 = A[0] ^ A[5];
    tt3 = A[10] ^ A[15];
    tt0 ^= A[20];
    tt2 ^= tt3;
    t1 = tt0 ^ tt2;

    tt0 = A[3] ^ A[8];
    tt1 = A[13] ^ A[18];
    tt0 ^= A[23] ^ tt1;
    tt0 = (tt0 << 1) | (tt0 >>> 63);
```

```
tt2 = A[1] ^ A[6];
tt3 = A[11] ^ A[16];
tt0 ^= A[21];
tt2 ^= tt3;
t2 = tt0 ^ tt2;

tt0 = A[4] ^ A[9];
tt1 = A[14] ^ A[19];
tt0 ^= A[24] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[2] ^ A[7];
tt3 = A[12] ^ A[17];
tt0 ^= A[22];
tt2 ^= tt3;
t3 = tt0 ^ tt2;

tt0 = A[0] ^ A[5];
tt1 = A[10] ^ A[15];
tt0 ^= A[20] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[3] ^ A[8];
tt3 = A[13] ^ A[18];
tt0 ^= A[23];
tt2 ^= tt3;
t4 = tt0 ^ tt2;

A[0] = A[0] ^ t0;
A[5] = A[5] ^ t0;
A[10] = A[10] ^ t0;
A[15] = A[15] ^ t0;
A[20] = A[20] ^ t0;
A[1] = A[1] ^ t1;
A[6] = A[6] ^ t1;
A[11] = A[11] ^ t1;
A[16] = A[16] ^ t1;
A[21] = A[21] ^ t1;
A[2] = A[2] ^ t2;
A[7] = A[7] ^ t2;
A[12] = A[12] ^ t2;
A[17] = A[17] ^ t2;
A[22] = A[22] ^ t2;
A[3] = A[3] ^ t3;
```

```
A[8] = A[8] ^ t3;
A[13] = A[13] ^ t3;
A[18] = A[18] ^ t3;
A[23] = A[23] ^ t3;
A[4] = A[4] ^ t4;
A[9] = A[9] ^ t4;
A[14] = A[14] ^ t4;
A[19] = A[19] ^ t4;
A[24] = A[24] ^ t4;
A[5] = (A[5] << 36) | (A[5] >>> (64 - 36));
A[10] = (A[10] << 3) | (A[10] >>> (64 - 3));
A[15] = (A[15] << 41) | (A[15] >>> (64 - 41));
A[20] = (A[20] << 18) | (A[20] >>> (64 - 18));
A[1] = (A[1] << 1) | (A[1] >>> (64 - 1));
A[6] = (A[6] << 44) | (A[6] >>> (64 - 44));
A[11] = (A[11] << 10) | (A[11] >>> (64 - 10));
A[16] = (A[16] << 45) | (A[16] >>> (64 - 45));
A[21] = (A[21] << 2) | (A[21] >>> (64 - 2));
A[2] = (A[2] << 62) | (A[2] >>> (64 - 62));
A[7] = (A[7] << 6) | (A[7] >>> (64 - 6));
A[12] = (A[12] << 43) | (A[12] >>> (64 - 43));
A[17] = (A[17] << 15) | (A[17] >>> (64 - 15));
A[22] = (A[22] << 61) | (A[22] >>> (64 - 61));
A[3] = (A[3] << 28) | (A[3] >>> (64 - 28));
A[8] = (A[8] << 55) | (A[8] >>> (64 - 55));
A[13] = (A[13] << 25) | (A[13] >>> (64 - 25));
A[18] = (A[18] << 21) | (A[18] >>> (64 - 21));
A[23] = (A[23] << 56) | (A[23] >>> (64 - 56));
A[4] = (A[4] << 27) | (A[4] >>> (64 - 27));
A[9] = (A[9] << 20) | (A[9] >>> (64 - 20));
A[14] = (A[14] << 39) | (A[14] >>> (64 - 39));
A[19] = (A[19] << 8) | (A[19] >>> (64 - 8));
A[24] = (A[24] << 14) | (A[24] >>> (64 - 14));
bnn = ~A[12];
kt = A[6] | A[12];
c0 = A[0] ^ kt;
kt = bnn | A[18];
c1 = A[6] ^ kt;
kt = A[18] & A[24];
c2 = A[12] ^ kt;
kt = A[24] | A[0];
c3 = A[18] ^ kt;
```

```
kt = A[0] & A[6];
c4 = A[24] ^ kt;
A[0] = c0;
A[6] = c1;
A[12] = c2;
A[18] = c3;
A[24] = c4;
bnn = ~A[22];
kt = A[9] | A[10];
c0 = A[3] ^ kt;
kt = A[10] & A[16];
c1 = A[9] ^ kt;
kt = A[16] | bnn;
c2 = A[10] ^ kt;
kt = A[22] | A[3];
c3 = A[16] ^ kt;
kt = A[3] & A[9];
c4 = A[22] ^ kt;
A[3] = c0;
A[9] = c1;
A[10] = c2;
A[16] = c3;
A[22] = c4;
bnn = ~A[19];
kt = A[7] | A[13];
c0 = A[1] ^ kt;
kt = A[13] & A[19];
c1 = A[7] ^ kt;
kt = bnn & A[20];
c2 = A[13] ^ kt;
kt = A[20] | A[1];
c3 = bnn ^ kt;
kt = A[1] & A[7];
c4 = A[20] ^ kt;
A[1] = c0;
A[7] = c1;
A[13] = c2;
A[19] = c3;
A[20] = c4;
bnn = ~A[17];
kt = A[5] & A[11];
c0 = A[4] ^ kt;
```

```
kt = A[11] | A[17];
c1 = A[5] ^ kt;
kt = bnn | A[23];
c2 = A[11] ^ kt;
kt = A[23] & A[4];
c3 = bnn ^ kt;
kt = A[4] | A[5];
c4 = A[23] ^ kt;
A[4] = c0;
A[5] = c1;
A[11] = c2;
A[17] = c3;
A[23] = c4;
bnn = ~A[8];
kt = bnn & A[14];
c0 = A[2] ^ kt;
kt = A[14] | A[15];
c1 = bnn ^ kt;
kt = A[15] & A[21];
c2 = A[14] ^ kt;
kt = A[21] | A[2];
c3 = A[15] ^ kt;
kt = A[2] & A[8];
c4 = A[21] ^ kt;
A[2] = c0;
A[8] = c1;
A[14] = c2;
A[15] = c3;
A[21] = c4;
A[0] = A[0] ^ RC[j + 0];

tt0 = A[6] ^ A[9];
tt1 = A[7] ^ A[5];
tt0 ^= A[8] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[24] ^ A[22];
tt3 = A[20] ^ A[23];
tt0 ^= A[21];
tt2 ^= tt3;
t0 = tt0 ^ tt2;

tt0 = A[12] ^ A[10];
```

```
tt1 = A[13] ^ A[11];
tt0 ^= A[14] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[0] ^ A[3];
tt3 = A[1] ^ A[4];
tt0 ^= A[2];
tt2 ^= tt3;
t1 = tt0 ^ tt2;

tt0 = A[18] ^ A[16];
tt1 = A[19] ^ A[17];
tt0 ^= A[15] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[6] ^ A[9];
tt3 = A[7] ^ A[5];
tt0 ^= A[8];
tt2 ^= tt3;
t2 = tt0 ^ tt2;

tt0 = A[24] ^ A[22];
tt1 = A[20] ^ A[23];
tt0 ^= A[21] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[12] ^ A[10];
tt3 = A[13] ^ A[11];
tt0 ^= A[14];
tt2 ^= tt3;
t3 = tt0 ^ tt2;

tt0 = A[0] ^ A[3];
tt1 = A[1] ^ A[4];
tt0 ^= A[2] ^ tt1;
tt0 = (tt0 << 1) | (tt0 >>> 63);
tt2 = A[18] ^ A[16];
tt3 = A[19] ^ A[17];
tt0 ^= A[15];
tt2 ^= tt3;
t4 = tt0 ^ tt2;

A[0] = A[0] ^ t0;
A[3] = A[3] ^ t0;
A[1] = A[1] ^ t0;
```

```
A[4] = A[4] ^ t0;
A[2] = A[2] ^ t0;
A[6] = A[6] ^ t1;
A[9] = A[9] ^ t1;
A[7] = A[7] ^ t1;
A[5] = A[5] ^ t1;
A[8] = A[8] ^ t1;
A[12] = A[12] ^ t2;
A[10] = A[10] ^ t2;
A[13] = A[13] ^ t2;
A[11] = A[11] ^ t2;
A[14] = A[14] ^ t2;
A[18] = A[18] ^ t3;
A[16] = A[16] ^ t3;
A[19] = A[19] ^ t3;
A[17] = A[17] ^ t3;
A[15] = A[15] ^ t3;
A[24] = A[24] ^ t4;
A[22] = A[22] ^ t4;
A[20] = A[20] ^ t4;
A[23] = A[23] ^ t4;
A[21] = A[21] ^ t4;
A[3] = (A[3] << 36) | (A[3] >>> (64 - 36));
A[1] = (A[1] << 3) | (A[1] >>> (64 - 3));
A[4] = (A[4] << 41) | (A[4] >>> (64 - 41));
A[2] = (A[2] << 18) | (A[2] >>> (64 - 18));
A[6] = (A[6] << 1) | (A[6] >>> (64 - 1));
A[9] = (A[9] << 44) | (A[9] >>> (64 - 44));
A[7] = (A[7] << 10) | (A[7] >>> (64 - 10));
A[5] = (A[5] << 45) | (A[5] >>> (64 - 45));
A[8] = (A[8] << 2) | (A[8] >>> (64 - 2));
A[12] = (A[12] << 62) | (A[12] >>> (64 - 62));
A[10] = (A[10] << 6) | (A[10] >>> (64 - 6));
A[13] = (A[13] << 43) | (A[13] >>> (64 - 43));
A[11] = (A[11] << 15) | (A[11] >>> (64 - 15));
A[14] = (A[14] << 61) | (A[14] >>> (64 - 61));
A[18] = (A[18] << 28) | (A[18] >>> (64 - 28));
A[16] = (A[16] << 55) | (A[16] >>> (64 - 55));
A[19] = (A[19] << 25) | (A[19] >>> (64 - 25));
A[17] = (A[17] << 21) | (A[17] >>> (64 - 21));
A[15] = (A[15] << 56) | (A[15] >>> (64 - 56));
A[24] = (A[24] << 27) | (A[24] >>> (64 - 27));
```

```
A[22] = (A[22] << 20) | (A[22] >>> (64 - 20));
A[20] = (A[20] << 39) | (A[20] >>> (64 - 39));
A[23] = (A[23] << 8) | (A[23] >>> (64 - 8));
A[21] = (A[21] << 14) | (A[21] >>> (64 - 14));
bnn = ~A[13];
kt = A[9] | A[13];
c0 = A[0] ^ kt;
kt = bnn | A[17];
c1 = A[9] ^ kt;
kt = A[17] & A[21];
c2 = A[13] ^ kt;
kt = A[21] | A[0];
c3 = A[17] ^ kt;
kt = A[0] & A[9];
c4 = A[21] ^ kt;
A[0] = c0;
A[9] = c1;
A[13] = c2;
A[17] = c3;
A[21] = c4;
bnn = ~A[14];
kt = A[22] | A[1];
c0 = A[18] ^ kt;
kt = A[1] & A[5];
c1 = A[22] ^ kt;
kt = A[5] | bnn;
c2 = A[1] ^ kt;
kt = A[14] | A[18];
c3 = A[5] ^ kt;
kt = A[18] & A[22];
c4 = A[14] ^ kt;
A[18] = c0;
A[22] = c1;
A[1] = c2;
A[5] = c3;
A[14] = c4;
bnn = ~A[23];
kt = A[10] | A[19];
c0 = A[6] ^ kt;
kt = A[19] & A[23];
c1 = A[10] ^ kt;
kt = bnn & A[2];
```

```
c2 = A[19] ^ kt;
kt = A[2] | A[6];
c3 = bnn ^ kt;
kt = A[6] & A[10];
c4 = A[2] ^ kt;
A[6] = c0;
A[10] = c1;
A[19] = c2;
A[23] = c3;
A[2] = c4;
bnn = ~A[11];
kt = A[3] & A[7];
c0 = A[24] ^ kt;
kt = A[7] | A[11];
c1 = A[3] ^ kt;
kt = bnn | A[15];
c2 = A[7] ^ kt;
kt = A[15] & A[24];
c3 = bnn ^ kt;
kt = A[24] | A[3];
c4 = A[15] ^ kt;
A[24] = c0;
A[3] = c1;
A[7] = c2;
A[11] = c3;
A[15] = c4;
bnn = ~A[16];
kt = bnn & A[20];
c0 = A[12] ^ kt;
kt = A[20] | A[4];
c1 = bnn ^ kt;
kt = A[4] & A[8];
c2 = A[20] ^ kt;
kt = A[8] | A[12];
c3 = A[4] ^ kt;
kt = A[12] & A[16];
c4 = A[8] ^ kt;
A[12] = c0;
A[16] = c1;
A[20] = c2;
A[4] = c3;
A[8] = c4;
```

```
        A[0] = A[0] ^ RC[j + 1];
        t = A[5];
        A[5] = A[18];
        A[18] = A[11];
        A[11] = A[10];
        A[10] = A[6];
        A[6] = A[22];
        A[22] = A[20];
        A[20] = A[12];
        A[12] = A[19];
        A[19] = A[15];
        A[15] = A[24];
        A[24] = A[8];
        A[8] = t;
        t = A[1];
        A[1] = A[9];
        A[9] = A[14];
        A[14] = A[2];
        A[2] = A[13];
        A[13] = A[23];
        A[23] = A[4];
        A[4] = A[21];
        A[21] = A[16];
        A[16] = A[3];
        A[3] = A[17];
        A[17] = A[7];
        A[7] = t;
    }
}

/**
 * @see org.ethereum.crypto.cryptohash.DigestEngine
 */
@Override
protected void doPadding(byte[] out, int off) {
    int ptr = flush();
    byte[] buf = getBlockBuffer();
    if ((ptr + 1) == buf.length) {
        buf[ptr] = (byte) 0x81;
    } else {
        buf[ptr] = (byte) 0x01;
        for (int i = ptr + 1; i < (buf.length - 1); i++) {
```

```
            buf[i] = 0;
        }
        buf[buf.length - 1] = (byte) 0x80;
    }
    processBlock(buf);
    A[1] = ~A[1];
    A[2] = ~A[2];
    A[8] = ~A[8];
    A[12] = ~A[12];
    A[17] = ~A[17];
    A[20] = ~A[20];
    int dlen = engineGetDigestLength();
    for (int i = 0; i < dlen; i += 8) {
        encodeLELong(A[i >>> 3], tmpOut, i);
    }
    System.arraycopy(tmpOut, 0, out, off, dlen);
}

/**
 * @see org.ethereum.crypto.cryptohash.DigestEngine
 */
@Override
protected void doInit() {
    A = new long[25];
    tmpOut = new byte[(engineGetDigestLength() + 7) & ~7];
    doReset();
}

/**
 * @see org.ethereum.crypto.cryptohash.Digest
 */
@Override
public int getBlockLength() {
    return 200 - 2 * engineGetDigestLength();
}

private final void doReset() {
    for (int i = 0; i < 25; i++) {
        A[i] = 0;
    }
    A[1] = 0xFFFFFFFFFFFFFFFFL;
    A[2] = 0xFFFFFFFFFFFFFFFFL;
```

```
        A[8] = 0xFFFFFFFFFFFFFFFFL;
        A[12] = 0xFFFFFFFFFFFFFFFFL;
        A[17] = 0xFFFFFFFFFFFFFFFFL;
        A[20] = 0xFFFFFFFFFFFFFFFFL;
    }

    /**
     * @see org.ethereum.crypto.cryptohash.DigestEngine
     */
    protected Digest copyState(KeccakCore dst) {
        System.arraycopy(A, 0, dst.A, 0, 25);
        return super.copyState(dst);
    }

    /**
     * @see org.ethereum.crypto.cryptohash.Digest
     */
    @Override
    public String toString() {
        return "Keccak-" + (engineGetDigestLength() << 3);
    }
}
```

133:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\HashUtil.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.spongycastle.crypto.Digest;
import org.spongycastle.crypto.digests.RIPEMD160Digest;
import org.spongycastle.util.encoders.Hex;

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.Provider;
import java.security.Security;
import java.util.Random;

import static java.util.Arrays.copyOfRange;
import static org.ethereum.util.ByteUtil.EMPTY_BYTE_ARRAY;

public class HashUtil {
```

```
    private static final Logger LOG = LoggerFactory.getLogger(HashUtil.class);

    public static final byte[] EMPTY_DATA_HASH;
    public static final byte[] EMPTY_LIST_HASH;
    public static final byte[] EMPTY_TRIE_HASH;

    private static final Provider CRYPTO_PROVIDER;

    private static final String HASH_256_ALGORITHM_NAME;
    private static final String HASH_512_ALGORITHM_NAME;

    static {
        SystemProperties props = SystemProperties.getDefault();
        Security.addProvider(SpongyCastleProvider.getInstance());
        CRYPTO_PROVIDER = Security.getProvider(props.getCryptoProviderName());
        HASH_256_ALGORITHM_NAME = props.getHash256AlgName();
        HASH_512_ALGORITHM_NAME = props.getHash512AlgName();
        EMPTY_DATA_HASH = sha3(EMPTY_BYTE_ARRAY);
        EMPTY_LIST_HASH = sha3(RLP.encodeList());
        EMPTY_TRIE_HASH = sha3(RLP.encodeElement(EMPTY_BYTE_ARRAY));
    }

    /**
     * @param input - data for hashing
     * @return - sha256 hash of the data
     */
    public static byte[] sha256(byte[] input) {
        try {
            MessageDigest sha256digest = MessageDigest.getInstance("SHA-256");
            return sha256digest.digest(input);
        } catch (NoSuchAlgorithmException e) {
            LOG.error("Can't find such algorithm", e);
            throw new RuntimeException(e);
        }
    }

    public static byte[] sha3(byte[] input) {
        MessageDigest digest;
        try {
            digest = MessageDigest.getInstance(HASH_256_ALGORITHM_NAME,
CRYPTO_PROVIDER);
```

```
            digest.update(input);
            return digest.digest();
        } catch (NoSuchAlgorithmException e) {
            LOG.error("Can't find such algorithm", e);
            throw new RuntimeException(e);
        }

    }

    public static byte[] sha3(byte[] input1, byte[] input2) {
        MessageDigest digest;
        try {
            digest = MessageDigest.getInstance(HASH_256_ALGORITHM_NAME,
CRYPTO_PROVIDER);
            digest.update(input1, 0, input1.length);
            digest.update(input2, 0, input2.length);
            return digest.digest();
        } catch (NoSuchAlgorithmException e) {
            LOG.error("Can't find such algorithm", e);
            throw new RuntimeException(e);
        }
    }

    /**
     * hashing chunk of the data
     *
     * @param input  - data for hash
     * @param start  - start of hashing chunk
     * @param length - length of hashing chunk
     * @return - keccak hash of the chunk
     */
    public static byte[] sha3(byte[] input, int start, int length) {
        MessageDigest digest;
        try {
            digest = MessageDigest.getInstance(HASH_256_ALGORITHM_NAME,
CRYPTO_PROVIDER);
            digest.update(input, start, length);
            return digest.digest();
        } catch (NoSuchAlgorithmException e) {
            LOG.error("Can't find such algorithm", e);
            throw new RuntimeException(e);
        }
```

```
    }

    public static byte[] sha512(byte[] input) {
        MessageDigest digest;
        try {
            digest = MessageDigest.getInstance(HASH_512_ALGORITHM_NAME,
CRYPTO_PROVIDER);
            digest.update(input);
            return digest.digest();
        } catch (NoSuchAlgorithmException e) {
            LOG.error("Can't find such algorithm", e);
            throw new RuntimeException(e);
        }
    }

    /**
     * @param data - message to hash
     * @return - reipmd160 hash of the message
     */
    public static byte[] ripemd160(byte[] data) {
        Digest digest = new RIPEMD160Digest();
        if (data != null) {
            byte[] resBuf = new byte[digest.getDigestSize()];
            digest.update(data, 0, data.length);
            digest.doFinal(resBuf, 0);
            return resBuf;
        }
        throw new NullPointerException("Can't hash a NULL value");
    }

    /**
     * Calculates RIGTMOST160(SHA3(input)). This is used in address
     * calculations. *
     *
     * @param input - data
     * @return - 20 right bytes of the hash keccak of the data
     */
    public static byte[] sha3omit12(byte[] input) {
        byte[] hash = sha3(input);
        return copyOfRange(hash, 12, hash.length);
    }
```

```
    /**
     * The way to calculate new address inside ethereum
     *
     * @param addr  - creating address
     * @param nonce - nonce of creating address
     * @return new address
     */
    public static byte[] calcNewAddr(byte[] addr, byte[] nonce) {

        byte[] encSender = RLP.encodeElement(addr);
        byte[] encNonce = RLP.encodeBigInteger(new BigInteger(1, nonce));

        return sha3omit12(RLP.encodeList(encSender, encNonce));
    }

    /**
     * The way to calculate new address inside ethereum for {@link
org.ethereum.vm.OpCode#CREATE2}
     * sha3(0xff ++ msg.sender ++ salt ++ sha3(init_code)))[12:]
     *
     * @param senderAddr - creating address
     * @param initCode   - contract init code
     * @param salt       - salt to make different result addresses
     * @return new address
     */
    public static byte[] calcSaltAddr(byte[] senderAddr, byte[] initCode, byte[] salt) {
        // 1 - 0xff length, 32 bytes - keccak-256
        byte[] data = new byte[1 + senderAddr.length + salt.length + 32];
        data[0] = (byte) 0xff;
        int currentOffset = 1;
        System.arraycopy(senderAddr, 0, data, currentOffset, senderAddr.length);
        currentOffset += senderAddr.length;
        System.arraycopy(salt, 0, data, currentOffset, salt.length);
        currentOffset += salt.length;
        byte[] sha3InitCode = sha3(initCode);
        System.arraycopy(sha3InitCode, 0, data, currentOffset, sha3InitCode.length);

        return sha3omit12(data);
    }

    /**
     * @param input -
```

```
     * @return -
     * @see #doubleDigest(byte[], int, int)
     */
    public static byte[] doubleDigest(byte[] input) {
        return doubleDigest(input, 0, input.length);
    }

    /**
     * Calculates the SHA-256 hash of the given byte range, and then hashes the
     * resulting hash again. This is standard procedure in Bitcoin. The
     * resulting hash is in big endian form.
     *
     * @param input  -
     * @param offset -
     * @param length -
     * @return -
     */
    public static byte[] doubleDigest(byte[] input, int offset, int length) {
        try {
            MessageDigest sha256digest = MessageDigest.getInstance("SHA-256");
            sha256digest.reset();
            sha256digest.update(input, offset, length);
            byte[] first = sha256digest.digest();
            return sha256digest.digest(first);
        } catch (NoSuchAlgorithmException e) {
            LOG.error("Can't find such algorithm", e);
            throw new RuntimeException(e);
        }
    }

    /**
     * @return generates random peer id for the HelloMessage
     */
    public static byte[] randomPeerId() {

        byte[] peerIdBytes = new BigInteger(512, Utils.getRandom()).toByteArray();

        final String peerId;
        if (peerIdBytes.length > 64) {
            peerId = Hex.toHexString(peerIdBytes, 1, 64);
        } else {
            peerId = Hex.toHexString(peerIdBytes);
```

```
        }

        return Hex.decode(peerId);
    }

    /**
     * @return - generate random 32 byte hash
     */
    public static byte[] randomHash() {

        byte[] randomHash = new byte[32];
        Random random = new Random();
        random.nextBytes(randomHash);
        return randomHash;
    }

    public static String shortHash(byte[] hash) {
        return Hex.toHexString(hash).substring(0, 6);
    }
}
```

134:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\jce\ECAlgorithmParameters.java

```
import java.security.spec.ECParameterSpec;
import java.security.spec.InvalidParameterSpecException;

public final class ECAlgorithmParameters {

    public static final String ALGORITHM = "EC";
    public static final String CURVE_NAME = "secp256k1";

    private ECAlgorithmParameters() {
    }

    private static class Holder {
        private static final AlgorithmParameters INSTANCE;

        private static final ECGenParameterSpec SECP256K1_CURVE
                = new ECGenParameterSpec(CURVE_NAME);

        static {
            try {
```

```
            INSTANCE = AlgorithmParameters.getInstance(ALGORITHM);
            INSTANCE.init(SECP256K1_CURVE);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(
                    "Assumed the JRE supports EC algorithm params", ex);
        } catch (InvalidParameterSpecException ex) {
            throw new AssertionError(
                    "Assumed correct key spec statically", ex);
        }
    }
}

public static ECParameterSpec getParameterSpec() {
    try {
        return Holder.INSTANCE.getParameterSpec(ECParameterSpec.class);
    } catch (InvalidParameterSpecException ex) {
        throw new AssertionError(
                "Assumed correct key spec statically", ex);
    }
}

public static byte[] getASN1Encoding() {
    try {
        return Holder.INSTANCE.getEncoded();
    } catch (IOException ex) {
        throw new AssertionError(
                "Assumed algo params has been initialized", ex);
    }
}
}
```

135:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\jce\ECKeyAgreement.java

```
public final class ECKeyAgreement {

    public static final String ALGORITHM = "ECDH";

    private static final String algorithmAssertionMsg =
            "Assumed the JRE supports EC key agreement";

    private ECKeyAgreement() {
```

```
    }

    public static KeyAgreement getInstance() {
        try {
            return KeyAgreement.getInstance(ALGORITHM);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        }
    }

    public static KeyAgreement getInstance(final String provider) throws NoSuchProviderException
{
        try {
            return KeyAgreement.getInstance(ALGORITHM, provider);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        }
    }

    public static KeyAgreement getInstance(final Provider provider) {
        try {
            return KeyAgreement.getInstance(ALGORITHM, provider);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        }
    }
}
```

136:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\jce\ECKeyFactory.java

```
public final class ECKeyFactory {

    public static final String ALGORITHM = "EC";

    private static final String algorithmAssertionMsg =
            "Assumed the JRE supports EC key factories";

    private ECKeyFactory() {
    }

    private static class Holder {
```

```
        private static final KeyFactory INSTANCE;

        static {
            try {
                INSTANCE = KeyFactory.getInstance(ALGORITHM);
            } catch (NoSuchAlgorithmException ex) {
                throw new AssertionError(algorithmAssertionMsg, ex);
            }
        }
    }

    public static KeyFactory getInstance() {
        return Holder.INSTANCE;
    }

    public static KeyFactory getInstance(final String provider) throws NoSuchProviderException {
        try {
            return KeyFactory.getInstance(ALGORITHM, provider);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        }
    }

    public static KeyFactory getInstance(final Provider provider) {
        try {
            return KeyFactory.getInstance(ALGORITHM, provider);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        }
    }
}
```

137:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\jce\ECKeyPairGenerator.java

```
    public static final String ALGORITHM = "EC";
    public static final String CURVE_NAME = "secp256k1";

    private static final String algorithmAssertionMsg =
            "Assumed JRE supports EC key pair generation";

    private static final String keySpecAssertionMsg =
```

```
            "Assumed correct key spec statically";

    private static final ECGenParameterSpec SECP256K1_CURVE
            = new ECGenParameterSpec(CURVE_NAME);

    private ECKeyPairGenerator() {
    }

    private static class Holder {
        private static final KeyPairGenerator INSTANCE;

        static {
            try {
                INSTANCE = KeyPairGenerator.getInstance(ALGORITHM);
                INSTANCE.initialize(SECP256K1_CURVE);
            } catch (NoSuchAlgorithmException ex) {
                throw new AssertionError(algorithmAssertionMsg, ex);
            } catch (InvalidAlgorithmParameterException ex) {
                throw new AssertionError(keySpecAssertionMsg, ex);
            }
        }
    }

    public static KeyPair generateKeyPair() {
        return Holder.INSTANCE.generateKeyPair();
    }

    public static KeyPairGenerator getInstance(final String provider, final SecureRandom random)
throws NoSuchProviderException {
        try {
            final KeyPairGenerator gen = KeyPairGenerator.getInstance(ALGORITHM, provider);
            gen.initialize(SECP256K1_CURVE, random);
            return gen;
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        } catch (InvalidAlgorithmParameterException ex) {
            throw new AssertionError(keySpecAssertionMsg, ex);
        }
    }

    public static KeyPairGenerator getInstance(final Provider provider, final SecureRandom
random) {
```

```
        try {
            final KeyPairGenerator gen = KeyPairGenerator.getInstance(ALGORITHM, provider);
            gen.initialize(SECP256K1_CURVE, random);
            return gen;
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(algorithmAssertionMsg, ex);
        } catch (InvalidAlgorithmParameterException ex) {
            throw new AssertionError(keySpecAssertionMsg, ex);
        }
    }
}
```

138:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\jce\ECSignatureFactory.java

```
public final class ECSignatureFactory {

    public static final String RAW_ALGORITHM = "NONEwithECDSA";

    private static final String rawAlgorithmAssertionMsg =
            "Assumed the JRE supports NONEwithECDSA signatures";

    private ECSignatureFactory() {
    }

    public static Signature getRawInstance() {
        try {
            return Signature.getInstance(RAW_ALGORITHM);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(rawAlgorithmAssertionMsg, ex);
        }
    }

    public static Signature getRawInstance(final String provider) throws NoSuchProviderException {
        try {
            return Signature.getInstance(RAW_ALGORITHM, provider);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(rawAlgorithmAssertionMsg, ex);
        }
    }

    public static Signature getRawInstance(final Provider provider) {
```

```
        try {
            return Signature.getInstance(RAW_ALGORITHM, provider);
        } catch (NoSuchAlgorithmException ex) {
            throw new AssertionError(rawAlgorithmAssertionMsg, ex);
        }
    }
}
```

139:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\crypto\jce\SpongyCastleProvider.java

```
public final class SpongyCastleProvider {

    private static class Holder {
        private static final Provider INSTANCE;

        static {
            Provider p = Security.getProvider("SC");

            INSTANCE = (p != null) ? p : new BouncyCastleProvider();

            INSTANCE.put("MessageDigest.ETH-KECCAK-256",
"org.ethereum.crypto.cryptohash.Keccak256");
            INSTANCE.put("MessageDigest.ETH-KECCAK-512",
"org.ethereum.crypto.cryptohash.Keccak512");
        }
    }

    public static Provider getInstance() {
        return Holder.INSTANCE;
    }
}
```

140:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\AbstractCachedSource.java

```
 * Created by Anton Nashatyrev on 01.12.2016.
 */
public abstract class AbstractCachedSource<Key, Value>
        extends AbstractChainedSource<Key, Value, Key, Value>
        implements CachedSource<Key, Value> {

    private final Object lock = new Object();
```

```
/**
 * Like the Optional interface represents either the value cached
 * or null cached (i.e. cache knows that underlying storage contain null)
 */
public interface Entry<V> {
    V value();
}

static final class SimpleEntry<V> implements Entry<V> {
    private V val;

    public SimpleEntry(V val) {
        this.val = val;
    }

    @Override
    public V value() {
        return val;
    }
}

protected MemSizeEstimator<Key> keySizeEstimator;
protected MemSizeEstimator<Value> valueSizeEstimator;
private int size = 0;

public AbstractCachedSource(Source<Key, Value> source) {
    super(source);
}

/**
 * Returns the cached value if exist.
 * Method doesn't look into the underlying storage
 *
 * @return The value Entry if it cached (Entry may has null value if null value is cached),
 * or null if no information in the cache for this key
 */
abstract Entry<Value> getCached(Key key);

/**
 * Needs to be called by the implementation when cache entry is added
 * Only new entries should be accounted for accurate size tracking
```

```
     * If the value for the key is changed the {@link #cacheRemoved}
     * needs to be called first
     */
    protected void cacheAdded(Key key, Value value) {
        synchronized (lock) {
            if (keySizeEstimator != null) {
                size += keySizeEstimator.estimateSize(key);
            }
            if (valueSizeEstimator != null) {
                size += valueSizeEstimator.estimateSize(value);
            }
        }
    }

    /**
     * Needs to be called by the implementation when cache entry is removed
     */
    protected void cacheRemoved(Key key, Value value) {
        synchronized (lock) {
            if (keySizeEstimator != null) {
                size -= keySizeEstimator.estimateSize(key);
            }
            if (valueSizeEstimator != null) {
                size -= valueSizeEstimator.estimateSize(value);
            }
        }
    }

    /**
     * Needs to be called by the implementation when cache is cleared
     */
    protected void cacheCleared() {
        synchronized (lock) {
            size = 0;
        }
    }

    /**
     * Sets the key/value size estimators
     */
    public AbstractCachedSource<Key, Value> withSizeEstimators(MemSizeEstimator<Key>
keySizeEstimator, MemSizeEstimator<Value> valueSizeEstimator) {
```

```
        this.keySizeEstimator = keySizeEstimator;
        this.valueSizeEstimator = valueSizeEstimator;
        return this;
    }

    @Override
    public long estimateCacheSize() {
        return size;
    }
}
```

141:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\AbstractChainedSource.java

```
 * <p>
 * Created by Anton Nashatyrev on 06.12.2016.
 */
public abstract class AbstractChainedSource<Key, Value, SourceKey, SourceValue> implements Source<Key, Value> {

    private Source<SourceKey, SourceValue> source;
    protected boolean flushSource;

    /**
     * Intended for subclasses which wishes to initialize the source
     * later via {@link #setSource(Source)} method
     */
    protected AbstractChainedSource() {
    }

    public AbstractChainedSource(Source<SourceKey, SourceValue> source) {
        this.source = source;
    }

    /**
     * Intended for subclasses which wishes to initialize the source later
     */
    protected void setSource(Source<SourceKey, SourceValue> src) {
        source = src;
    }

    public Source<SourceKey, SourceValue> getSource() {
        return source;
```

```
    }

    public void setFlushSource(boolean flushSource) {
        this.flushSource = flushSource;
    }

    /**
     * Invokes {@link #flushImpl()} and does backing Source flush if required
     *
     * @return true if this or source flush did any changes
     */
    @Override
    public synchronized boolean flush() {
        boolean ret = flushImpl();
        if (flushSource) {
            ret |= getSource().flush();
        }
        return ret;
    }

    /**
     * Should be overridden to do actual source flush
     */
    protected abstract boolean flushImpl();
}
```

142:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\AsyncFlushable.java

```
 * <p>
 * Created by Anton Nashatyrev on 02.02.2017.
 */
public interface AsyncFlushable {

    /**
     * Flip the backing storage so the current state will be flushed
     * when call {@link #flushAsync()} and all the newer changes will
     * be collected to a new backing store and will be flushed only on
     * subsequent flush call
     * <p>
     * The method is intended to make consistent flush from several
     * sources. I.e. at some point all the related Sources are flipped
     * synchronously first (this doesn't consume any time normally) and then
```

```
     * are flushed asynchronously
     * <p>
     * This call may block until a previous flush is completed (if still in progress)
     *
     * @throws InterruptedException
     */
    void flipStorage() throws InterruptedException;

    /**
     * Does async flush, i.e. returns immediately while starts doing flush in a separate thread
     * This call may still block if the previous flush is not complete yet
     *
     * @return Future when the actual flush is complete
     */
    ListenableFuture<Boolean> flushAsync() throws InterruptedException;
}
```

143:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\AsyncWriteCache.java

```
import java.util.Collection;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

/**
 * Created by Anton Nashatyrev on 18.01.2017.
 */
public abstract class AsyncWriteCache<Key, Value> extends AbstractCachedSource<Key, Value>
implements AsyncFlushable {
    private static final Logger logger = LoggerFactory.getLogger("db");

    private static ListeningExecutorService flushExecutor = MoreExecutors.listeningDecorator(
            Executors.newFixedThreadPool(2, new
ThreadFactoryBuilder().setNameFormat("AsyncWriteCacheThread-%d").build()));

    protected volatile WriteCache<Key, Value> curCache;
    protected WriteCache<Key, Value> flushingCache;

    private ListenableFuture<Boolean> lastFlush = Futures.immediateFuture(false);
```

```
private final ReadWriteLock rwLock = new ReentrantReadWriteLock();
private final ALock rLock = new ALock(rwLock.readLock());
private final ALock wLock = new ALock(rwLock.writeLock());

private String name = "<null>";

public AsyncWriteCache(Source<Key, Value> source) {
    super(source);
    flushingCache = createCache(source);
    flushingCache.setFlushSource(true);
    curCache = createCache(flushingCache);
}

protected abstract WriteCache<Key, Value> createCache(Source<Key, Value> source);

@Override
public Collection<Key> getModified() {
    try (ALock l = rLock.lock()) {
        return curCache.getModified();
    }
}

@Override
public boolean hasModified() {
    try (ALock l = rLock.lock()) {
        return curCache.hasModified();
    }
}

@Override
public void put(Key key, Value val) {
    try (ALock l = rLock.lock()) {
        curCache.put(key, val);
    }
}

@Override
public void delete(Key key) {
    try (ALock l = rLock.lock()) {
        curCache.delete(key);
    }
}
```

```
@Override
public Value get(Key key) {
    try (ALock l = rLock.lock()) {
        return curCache.get(key);
    }
}

@Override
public synchronized boolean flush() {
    try {
        flipStorage();
        flushAsync();
        return flushingCache.hasModified();
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
}

@Override
Entry<Value> getCached(Key key) {
    return curCache.getCached(key);
}

@Override
public synchronized void flipStorage() throws InterruptedException {
    // if previous flush still running
    try {
        if (!lastFlush.isDone()) {
            logger.debug("AsyncWriteCache (" + name + "): waiting for previous flush to complete");
        }
        lastFlush.get();
    } catch (ExecutionException e) {
        throw new RuntimeException(e);
    }

    try (ALock l = wLock.lock()) {
        flushingCache.cache = curCache.cache;
        curCache = createCache(flushingCache);
    }
}
```

```
    @Override
    public synchronized ListenableFuture<Boolean> flushAsync() throws InterruptedException {
        logger.debug("AsyncWriteCache (" + name + "): flush submitted");
        lastFlush = flushExecutor.submit(() -> {
            logger.debug("AsyncWriteCache (" + name + "): flush started");
            long s = System.currentTimeMillis();
            boolean ret = flushingCache.flush();
            logger.debug("AsyncWriteCache (" + name + "): flush completed in " +
(System.currentTimeMillis() - s) + " ms");
            return ret;
        });
        return lastFlush;
    }

    @Override
    public long estimateCacheSize() {
        // 2.0 is upper cache size estimation to take into account there are two
        // caches may exist simultaneously up to doubling cache size
        return (long) (curCache.estimateCacheSize() * 2.0);
    }

    @Override
    protected synchronized boolean flushImpl() {
        return false;
    }

    public AsyncWriteCache<Key, Value> withName(String name) {
        this.name = name;
        return this;
    }
}

144:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\org\ethereum\datasource\BatchSource.java
 * The semantics of a batch update is up to implementation:
 * it can be just performance optimization or batch update
 * can be atomic or other.
 * <p>
 * Created by Anton Nashatyrev on 01.11.2016.
 */
public interface BatchSource<K, V> extends Source<K, V> {
```

```
    /**
     * Do batch update
     *
     * @param rows Normally this Map is treated just as a collection
     *             of key-value pairs and shouldn't conform to a normal
     *             Map contract. Though it is up to implementation to
     *             require passing specific Maps
     */
    void updateBatch(Map<K, V> rows);
}
```

145:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\BatchSourceWriter.java

```
 * Clue class between Source and BatchSource
 * <p>
 * Created by Anton Nashatyrev on 29.11.2016.
 */
public class BatchSourceWriter<Key, Value> extends AbstractChainedSource<Key, Value, Key, Value> {

    Map<Key, Value> buf = new HashMap<>();

    public BatchSourceWriter(BatchSource<Key, Value> src) {
        super(src);
    }

    private BatchSource<Key, Value> getBatchSource() {
        return (BatchSource<Key, Value>) getSource();
    }

    @Override
    public synchronized void delete(Key key) {
        buf.put(key, null);
    }

    @Override
    public synchronized void put(Key key, Value val) {
        buf.put(key, val);
    }

    @Override
    public Value get(Key key) {
```

```
        return getSource().get(key);
    }

    @Override
    public synchronized boolean flushImpl() {
        if (!buf.isEmpty()) {
            getBatchSource().updateBatch(buf);
            buf.clear();
            return true;
        } else {
            return false;
        }
    }
}
```

146:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\BloomedSource.java

```
/**
 * Special optimization when the majority of get requests to the slower underlying source
 * are targeted to missing entries. The BloomFilter handles most of these requests.
 * <p>
 * Created by Anton Nashatyrev on 16.01.2017.
 */
public class BloomedSource extends AbstractChainedSource<byte[], byte[], byte[], byte[]> {
    private final static Logger logger = LoggerFactory.getLogger("db");

    private byte[] filterKey = HashUtil.sha3("filterKey".getBytes());

    QuotientFilter filter;
    int hits = 0;
    int misses = 0;
    int falseMisses = 0;
    boolean dirty = false;
    int maxBloomSize;

    public BloomedSource(Source<byte[], byte[]> source, int maxBloomSize) {
        super(source);
        this.maxBloomSize = maxBloomSize;
        byte[] filterBytes = source.get(filterKey);
        if (filterBytes != null) {
            if (filterBytes.length > 0) {
                filter = QuotientFilter.deserialize(filterBytes);
```

```
            } else {
                // filter size exceeded limit and is disabled forever
                filter = null;
            }
        } else {
            if (maxBloomSize > 0) {
                filter = QuotientFilter.create(50_000_000, 100_000);
            } else {
                // we can't re-enable filter later
                getSource().put(filterKey, new byte[0]);
            }
        }
//
//        new Thread() {
//            @Override
//            public void run() {
//                while(true) {
//                    synchronized (BloomedSource.this) {
//                        logger.debug("BloomedSource: hits: " + hits + ", misses: " + misses + ", false: " +
falseMisses);
//                        hits = misses = falseMisses = 0;
//                    }
//
//                    try {
//                        Thread.sleep(5000);
//                    } catch (InterruptedException e) {}
//                }
//            }
//        }.start();
    }

    public void startBlooming(QuotientFilter filter) {
        this.filter = filter;
    }

    public void stopBlooming() {
        filter = null;
    }

    @Override
    public void put(byte[] key, byte[] val) {
        if (filter != null) {
```

```
            filter.insert(key);
            dirty = true;
            if (filter.getAllocatedBytes() > maxBloomSize) {
                logger.debug("Bloom filter became too large (" + filter.getAllocatedBytes() + " exceeds
max threshold " + maxBloomSize + ") and is now disabled forever.");
                getSource().put(filterKey, new byte[0]);
                filter = null;
                dirty = false;
            }
        }
        getSource().put(key, val);
    }

    @Override
    public byte[] get(byte[] key) {
        if (filter == null) {
            return getSource().get(key);
        }

        if (!filter.maybeContains(key)) {
            hits++;
            return null;
        } else {
            byte[] ret = getSource().get(key);
            if (ret == null) {
                falseMisses++;
            } else {
                misses++;
            }
            return ret;
        }
    }

    @Override
    public void delete(byte[] key) {
        if (filter != null) {
            filter.remove(key);
        }
        getSource().delete(key);
    }

    @Override
```

```
    protected boolean flushImpl() {
        if (filter != null && dirty) {
            getSource().put(filterKey, filter.serialize());
            dirty = false;
            return true;
        } else {
            return false;
        }
    }
}
```

147:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\BloomFilter.java

```
/**
 * The class mostly borrowed from http://github.com/magnuss/java-bloomfilter
 * <p>
 * Implementation of a Bloom-filter, as described here:
 * http://en.wikipedia.org/wiki/Bloom_filter
 * <p>
 * For updates and bugfixes, see http://github.com/magnuss/java-bloomfilter
 * <p>
 * Inspired by the SimpleBloomFilter-class written by Ian Clarke. This
 * implementation provides a more evenly distributed Hash-function by
 * using a proper digest instead of the Java RNG. Many of the changes
 * were proposed in comments in his blog:
 * http://blog.locut.us/2008/01/12/a-decent-stand-alone-java-bloom-filter-implementation/
 *
 * @author Magnus Skjegstad <magnus@skjegstad.com>
 */
public class BloomFilter implements Serializable {
    private BitSet bitset;
    private int bitSetSize;
    private double bitsPerElement;
    private int expectedNumberOfFilterElements; // expected (maximum) number of elements to be added
    private int numberOfAddedElements; // number of elements actually added to the Bloom filter
    private int k; // number of hash functions

    /**
     * Constructs an empty Bloom filter. The total length of the Bloom filter will be
     * c*n.
```

```
     *
     * @param c is the number of bits used per element.
     * @param n is the expected number of elements the filter will contain.
     * @param k is the number of hash functions used.
     */
    public BloomFilter(double c, int n, int k) {
        this.expectedNumberOfFilterElements = n;
        this.k = k;
        this.bitsPerElement = c;
        this.bitSetSize = (int) Math.ceil(c * n);
        numberOfAddedElements = 0;
        this.bitset = new BitSet(bitSetSize);
    }

    /**
     * Constructs an empty Bloom filter. The optimal number of hash functions (k) is estimated from the total size of the Bloom
     * and the number of expected elements.
     *
     * @param bitSetSize              defines how many bits should be used in total for the filter.
     * @param expectedNumberOElements defines the maximum number of elements the filter is expected to contain.
     */
    public BloomFilter(int bitSetSize, int expectedNumberOElements) {
        this(bitSetSize / (double) expectedNumberOElements,
                expectedNumberOElements,
                (int) Math.round((bitSetSize / (double) expectedNumberOElements) * Math.log(2.0)));
    }

    /**
     * Constructs an empty Bloom filter with a given false positive probability. The number of bits per
     * element and the number of hash functions is estimated
     * to match the false positive probability.
     *
     * @param falsePositiveProbability is the desired false positive probability.
     * @param expectedNumberOfElements is the expected number of elements in the Bloom filter.
     */
    public BloomFilter(double falsePositiveProbability, int expectedNumberOfElements) {
        this(Math.ceil(-(Math.log(falsePositiveProbability) / Math.log(2))) / Math.log(2), // c = k / ln(2)
                expectedNumberOfElements,
                (int) Math.ceil(-(Math.log(falsePositiveProbability) / Math.log(2)))); // k = ceil(-log_2(false prob.))
```

```
    }

    /**
     * Construct a new Bloom filter based on existing Bloom filter data.
     *
     * @param bitSetSize                    defines how many bits should be used for the filter.
     * @param expectedNumberOfFilterElements defines the maximum number of elements the filter is expected to contain.
     * @param actualNumberOfFilterElements   specifies how many elements have been inserted into the <code>filterData</code> BitSet.
     * @param filterData                    a BitSet representing an existing Bloom filter.
     */
    public BloomFilter(int bitSetSize, int expectedNumberOfFilterElements, int actualNumberOfFilterElements, BitSet filterData) {
        this(bitSetSize, expectedNumberOfFilterElements);
        this.bitset = filterData;
        this.numberOfAddedElements = actualNumberOfFilterElements;
    }

    /**
     * Compares the contents of two instances to see if they are equal.
     *
     * @param obj is the object to compare to.
     * @return True if the contents of the objects are equal.
     */
    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final BloomFilter other = (BloomFilter) obj;
        if (this.expectedNumberOfFilterElements != other.expectedNumberOfFilterElements) {
            return false;
        }
        if (this.k != other.k) {
            return false;
        }
        if (this.bitSetSize != other.bitSetSize) {
            return false;
```

```
    }
    if (this.bitset != other.bitset && (this.bitset == null || !this.bitset.equals(other.bitset))) {
        return false;
    }
    return true;
}

/**
 * Calculates a hash code for this class.
 *
 * @return hash code representing the contents of an instance of this class.
 */
@Override
public int hashCode() {
    int hash = 7;
    hash = 61 * hash + (this.bitset != null ? this.bitset.hashCode() : 0);
    hash = 61 * hash + this.expectedNumberOfFilterElements;
    hash = 61 * hash + this.bitSetSize;
    hash = 61 * hash + this.k;
    return hash;
}



/**
 * Calculates the expected probability of false positives based on
 * the number of expected filter elements and the size of the Bloom filter.
 * <br /><br />
 * The value returned by this method is the <i>expected</i> rate of false
 * positives, assuming the number of inserted elements equals the number of
 * expected elements. If the number of elements in the Bloom filter is less
 * than the expected value, the true probability of false positives will be lower.
 *
 * @return expected probability of false positives.
 */
public double expectedFalsePositiveProbability() {
    return getFalsePositiveProbability(expectedNumberOfFilterElements);
}

/**
 * Calculate the probability of a false positive given the specified
 * number of inserted elements.
 *
```

```
     * @param numberOfElements number of inserted elements.
     * @return probability of a false positive.
     */
    public double getFalsePositiveProbability(double numberOfElements) {
        // (1 - e^(-k * n / m)) ^ k
        return Math.pow((1 - Math.exp(-k * (double) numberOfElements
                / (double) bitSetSize)), k);

    }

    /**
     * Get the current probability of a false positive. The probability is calculated from
     * the size of the Bloom filter and the current number of elements added to it.
     *
     * @return probability of false positives.
     */
    public double getFalsePositiveProbability() {
        return getFalsePositiveProbability(numberOfAddedElements);
    }



    /**
     * Returns the value chosen for K.<br />
     * <br />
     * K is the optimal number of hash functions based on the size
     * of the Bloom filter and the expected number of inserted elements.
     *
     * @return optimal k.
     */
    public int getK() {
        return k;
    }

    /**
     * Sets all bits to false in the Bloom filter.
     */
    public synchronized void clear() {
        bitset.clear();
        numberOfAddedElements = 0;
    }
```

```
/**
 * Adds an array of bytes to the Bloom filter.
 *
 * @param bytes array of bytes to add to the Bloom filter.
 */
public synchronized void add(byte[] bytes) {
    int[] hashes = createHashes(bytes, k);
    for (int hash : hashes) {
        bitset.set(Math.abs(hash % bitSetSize), true);
    }
    numberOfAddedElements++;
}

private int[] createHashes(byte[] bytes, int k) {
    int[] ret = new int[k];
    if (bytes.length / 4 < k) {
        int[] maxHashes = new int[bytes.length / 4];
        ByteUtil.bytesToInts(bytes, maxHashes, false);
        for (int i = 0; i < ret.length; i++) {
            ret[i] = maxHashes[i % maxHashes.length];
        }
    } else {
        ByteUtil.bytesToInts(bytes, ret, false);
    }
    return ret;
}

/**
 * Returns true if the array of bytes could have been inserted into the Bloom filter.
 * Use getFalsePositiveProbability() to calculate the probability of this
 * being correct.
 *
 * @param bytes array of bytes to check.
 * @return true if the array could have been inserted into the Bloom filter.
 */
public synchronized boolean contains(byte[] bytes) {
    int[] hashes = createHashes(bytes, k);
    for (int hash : hashes) {
        if (!bitset.get(Math.abs(hash % bitSetSize))) {
            return false;
        }
    }
```

```
    return true;
}

/**
 * Read a single bit from the Bloom filter.
 *
 * @param bit the bit to read.
 * @return true if the bit is set, false if it is not.
 */
public synchronized boolean getBit(int bit) {
    return bitset.get(bit);
}

/**
 * Set a single bit in the Bloom filter.
 *
 * @param bit   is the bit to set.
 * @param value If true, the bit is set. If false, the bit is cleared.
 */
public synchronized void setBit(int bit, boolean value) {
    bitset.set(bit, value);
}

/**
 * Return the bit set used to store the Bloom filter.
 *
 * @return bit set representing the Bloom filter.
 */
public synchronized BitSet getBitSet() {
    return bitset;
}

/**
 * Returns the number of bits in the Bloom filter. Use count() to retrieve
 * the number of inserted elements.
 *
 * @return the size of the bitset used by the Bloom filter.
 */
public synchronized int size() {
    return this.bitSetSize;
}
```

```
    /**
     * Returns the number of elements added to the Bloom filter after it
     * was constructed or after clear() was called.
     *
     * @return number of elements added to the Bloom filter.
     */
    public synchronized int count() {
        return this.numberOfAddedElements;
    }

    /**
     * Returns the expected number of elements to be inserted into the filter.
     * This value is the same value as the one passed to the constructor.
     *
     * @return expected number of elements.
     */
    public int getExpectedNumberOfElements() {
        return expectedNumberOfFilterElements;
    }

    /**
     * Get expected number of bits per element when the Bloom filter is full. This value is set by the
constructor
     * when the Bloom filter is created. See also getBitsPerElement().
     *
     * @return expected number of bits per element.
     */
    public double getExpectedBitsPerElement() {
        return this.bitsPerElement;
    }

    /**
     * Get actual number of bits per element based on the number of elements that have currently
been inserted and the length
     * of the Bloom filter. See also getExpectedBitsPerElement().
     *
     * @return number of bits per element.
     */
    public double getBitsPerElement() {
        return this.bitSetSize / (double) numberOfAddedElements;
    }
}
```

148:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\CachedSource.java

```
 * <p>
 * Created by Anton Nashatyrev on 21.10.2016.
 */
public interface CachedSource<Key, Value> extends Source<Key, Value> {

    /**
     * @return The underlying Source
     */
    Source<Key, Value> getSource();

    /**
     * @return Modified entry keys if this is a write cache
     */
    Collection<Key> getModified();

    /**
     * @return indicates the cache has modified entries
     */
    boolean hasModified();

    /**
     * Estimates the size of cached entries in bytes.
     * This value shouldn't be precise size of Java objects
     *
     * @return cache size in bytes
     */
    long estimateCacheSize();

    /**
     * Just a convenient shortcut to the most popular Sources with byte[] key
     */
    interface BytesKey<Value> extends CachedSource<byte[], Value> {
    }
}
```

149:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\CountingBytesSource.java

```
import java.util.Arrays;
```

```
/**
 * 'Reference counting' Source. Unlike regular Source if an entry was
 * e.g. 'put' twice it is actually deleted when 'delete' is called twice
 * I.e. each put increments counter and delete decrements counter, the
 * entry is deleted when the counter becomes zero.
 * <p>
 * Please note that the counting mechanism makes sense only for
 * {@link HashedKeySource} like Sources when any taken key can correspond to
 * the only value
 * <p>
 * This Source is constrained to byte[] values only as the counter
 * needs to be encoded to the backing Source value as byte[]
 * <p>
 * Created by Anton Nashatyrev on 08.11.2016.
 */
public class CountingBytesSource extends AbstractChainedSource<byte[], byte[], byte[], byte[]>
        implements HashedKeySource<byte[], byte[]> {

    QuotientFilter filter;
    boolean dirty = false;
    private byte[] filterKey = HashUtil.sha3("countingStateFilter".getBytes());

    public CountingBytesSource(Source<byte[], byte[]> src) {
        this(src, false);

    }

    public CountingBytesSource(Source<byte[], byte[]> src, boolean bloom) {
        super(src);
        byte[] filterBytes = src.get(filterKey);
        if (bloom) {
            if (filterBytes != null) {
                filter = QuotientFilter.deserialize(filterBytes);
            } else {
                filter = QuotientFilter.create(5_000_000, 10_000);
            }
        }
    }

    @Override
    public void put(byte[] key, byte[] val) {
        if (val == null) {
```

```
            delete(key);
            return;
        }

        synchronized (this) {
            byte[] srcVal = getSource().get(key);
            int srcCount = decodeCount(srcVal);
            if (srcCount >= 1) {
                if (filter != null) {
                    filter.insert(key);
                }
                dirty = true;
            }
            getSource().put(key, encodeCount(val, srcCount + 1));
        }
    }

    @Override
    public byte[] get(byte[] key) {
        return decodeValue(getSource().get(key));
    }

    @Override
    public void delete(byte[] key) {
        synchronized (this) {
            int srcCount;
            byte[] srcVal = null;
            if (filter == null || filter.maybeContains(key)) {
                srcVal = getSource().get(key);
                srcCount = decodeCount(srcVal);
            } else {
                srcCount = 1;
            }
            if (srcCount > 1) {
                getSource().put(key, encodeCount(decodeValue(srcVal), srcCount - 1));
            } else {
                getSource().delete(key);
            }
        }
    }

    @Override
```

```
    protected boolean flushImpl() {
        if (filter != null && dirty) {
            byte[] filterBytes;
            synchronized (this) {
                filterBytes = filter.serialize();
            }
            getSource().put(filterKey, filterBytes);
            dirty = false;
            return true;
        } else {
            return false;
        }
    }

    /**
     * Extracts value from the backing Source counter + value byte array
     */
    protected byte[] decodeValue(byte[] srcVal) {
        return srcVal == null ? null : Arrays.copyOfRange(srcVal, RLP.decode(srcVal, 0).getPos(),
srcVal.length);
    }

    /**
     * Extracts counter from the backing Source counter + value byte array
     */
    protected int decodeCount(byte[] srcVal) {
        return srcVal == null ? 0 : ByteUtil.byteArrayToInt((byte[]) RLP.decode(srcVal,
0).getDecoded());
    }

    /**
     * Composes value and counter into backing Source value
     */
    protected byte[] encodeCount(byte[] val, int count) {
        return ByteUtil.merge(RLP.encodeInt(count), val);
    }
}

150:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\org\ethereum\datasource\CountingQuotientFilter.java
        this.FINGERPRINT_MASK = LOW_MASK(QUOTIENT_BITS + REMAINDER_BITS);
    }
```

```
    public static CountingQuotientFilter create(long largestNumberOfElements, long
startingElements) {
        QuotientFilter filter = QuotientFilter.create(largestNumberOfElements, startingElements);
        return new CountingQuotientFilter(filter.QUOTIENT_BITS, filter.REMAINDER_BITS);
    }

    @Override
    public synchronized void insert(long hash) {
        if (super.maybeContains(hash)) {
            addRef(hash);
        } else {
            super.insert(hash);
        }
    }

    @Override
    public synchronized void remove(long hash) {
        if (super.maybeContains(hash) && delRef(hash) < 0) {
            super.remove(hash);
        }
    }

    @Override
    protected long hash(byte[] bytes) {
        long hash = 1;
        for (byte b : bytes) {
            hash = 31 * hash + b;
        }
        return hash;
    }

    public synchronized int getCollisionNumber() {
        return counters.size();
    }

    public long getEntryNumber() {
        return entries;
    }

    public long getMaxInsertions() {
        return MAX_INSERTIONS;
```

```
    }

    private void addRef(long hash) {
        long fp = fingerprint(hash);
        Counter cnt = counters.get(fp);
        if (cnt == null) {
            counters.put(fp, new Counter());
        } else {
            cnt.refs++;
        }
    }

    private int delRef(long hash) {
        long fp = fingerprint(hash);
        Counter cnt = counters.get(fp);
        if (cnt == null) {
            return -1;
        }
        if (--cnt.refs < 1) {
            counters.remove(fp);
        }
        return cnt.refs;
    }

    private long fingerprint(long hash) {
        return hash & FINGERPRINT_MASK;
    }

    private static class Counter {
        int refs = 1;
    }
}
```

151:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\DataSourceArray.java

```
/**
 * Stores List structure in Source structure
 * <p>
 * Created by Anton Nashatyrev on 17.03.2016.
 */
public class DataSourceArray<V> extends AbstractList<V> {
```

```
private ObjectDataSource<V> src;
private static final byte[] SIZE_KEY = Hex.decode("FFFFFFFFFFFFFFFF");
private int size = -1;

public DataSourceArray(ObjectDataSource<V> src) {
    this.src = src;
}

public synchronized boolean flush() {
    return src.flush();
}

@Override
public synchronized V set(int idx, V value) {
    if (idx >= size()) {
        setSize(idx + 1);
    }
    src.put(ByteUtil.intToBytes(idx), value);
    return value;
}

@Override
public synchronized void add(int index, V element) {
    set(index, element);
}

@Override
public synchronized V remove(int index) {
    throw new RuntimeException("Not supported yet.");
}

@Override
public synchronized V get(int idx) {
    if (idx < 0 || idx >= size()) {
        throw new IndexOutOfBoundsException(idx + " > " + size);
    }
    return src.get(ByteUtil.intToBytes(idx));
}

@Override
public synchronized int size() {
    if (size < 0) {
```

```
            byte[] sizeBB = src.getSource().get(SIZE_KEY);
            size = sizeBB == null ? 0 : ByteUtil.byteArrayToInt(sizeBB);
        }
        return size;
    }

    private synchronized void setSize(int newSize) {
        size = newSize;
        src.getSource().put(SIZE_KEY, ByteUtil.intToBytes(newSize));
    }
}
```

152:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\DbSettings.java

```
 * @since 26.04.2018
 */
public class DbSettings {

    public static final DbSettings DEFAULT = new DbSettings()
            .withMaxThreads(1)
            .withMaxOpenFiles(32);

    int maxOpenFiles;
    int maxThreads;

    private DbSettings() {
    }

    public static DbSettings newInstance() {
        DbSettings settings = new DbSettings();
        settings.maxOpenFiles = DEFAULT.maxOpenFiles;
        settings.maxThreads = DEFAULT.maxThreads;
        return settings;
    }

    public int getMaxOpenFiles() {
        return maxOpenFiles;
    }

    public DbSettings withMaxOpenFiles(int maxOpenFiles) {
        this.maxOpenFiles = maxOpenFiles;
        return this;
```

```
    }

    public int getMaxThreads() {
        return maxThreads;
    }

    public DbSettings withMaxThreads(int maxThreads) {
        this.maxThreads = maxThreads;
        return this;
    }
}
```

153:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\DbSource.java

```
 */
public interface DbSource<V> extends BatchSource<byte[], V> {

    /**
     * Sets the DB name.
     * This could be the underlying DB table/dir name
     */
    void setName(String name);

    /**
     * @return DB name
     */
    String getName();

    /**
     * Initializes DB (open table, connection, etc)
     * with default {@link DbSettings#DEFAULT}
     */
    void init();

    /**
     * Initializes DB (open table, connection, etc)
     *
     * @param settings DB settings
     */
    void init(DbSettings settings);

    /**
```

```
     * @return true if DB connection is alive
     */
    boolean isAlive();

    /**
     * Closes the DB table/connection
     */
    void close();

    /**
     * @return DB keys if this option is available
     * @throws RuntimeException if the method is not supported
     */
    Set<byte[]> keys() throws RuntimeException;

    /**
     * Closes database, destroys its data and finally runs init()
     */
    void reset();

    /**
     * If supported, retrieves a value using a key prefix.
     * Prefix extraction is meant to be done on the implementing side.<br>
     *
     * @param key         a key for the lookup
     * @param prefixBytes prefix length in bytes
     * @return first value picked by prefix lookup over DB or null if there is no match
     * @throws RuntimeException if operation is not supported
     */
    V prefixLookup(byte[] key, int prefixBytes);
}
```

154:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\HashedKeySource.java

```
 * Normally the Key is the hash of the Value
 * Usually such kind of sources are Merkle Trie backing stores
 * <p>
 * Created by Anton Nashatyrev on 08.11.2016.
 */
public interface HashedKeySource<Key, Value> extends Source<Key, Value> {
}
```

155:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\org\ethereum\datasource\inmem\HashMapDB.java

```
import org.ethereum.util.FastByteComparisons;

import java.util.Map;
import java.util.Set;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

/**
 * Created by Anton Nashatyrev on 12.10.2016.
 */
public class HashMapDB<V> implements DbSource<V> {

    protected final Map<byte[], V> storage;

    protected ReadWriteLock rwLock = new ReentrantReadWriteLock();
    protected ALock readLock = new ALock(rwLock.readLock());
    protected ALock writeLock = new ALock(rwLock.writeLock());

    public HashMapDB() {
        this(new ByteArrayMap<V>());
    }

    public HashMapDB(ByteArrayMap<V> storage) {
        this.storage = storage;
    }

    @Override
    public void put(byte[] key, V val) {
        if (val == null) {
            delete(key);
        } else {
            try (ALock l = writeLock.lock()) {
                storage.put(key, val);
            }
        }
    }

    @Override
    public V get(byte[] key) {
        try (ALock l = readLock.lock()) {
```

```
        return storage.get(key);
    }
}

@Override
public void delete(byte[] key) {
    try (ALock l = writeLock.lock()) {
        storage.remove(key);
    }
}

@Override
public boolean flush() {
    return true;
}

@Override
public void setName(String name) {
}

@Override
public String getName() {
    return "in-memory";
}

@Override
public void init() {
}

@Override
public void init(DbSettings settings) {
}

@Override
public boolean isAlive() {
    return true;
}

@Override
public void close() {
}
```

```
    @Override
    public Set<byte[]> keys() {
        try (ALock l = readLock.lock()) {
            return getStorage().keySet();
        }
    }

    @Override
    public void reset() {
        try (ALock l = writeLock.lock()) {
            storage.clear();
        }
    }

    @Override
    public V prefixLookup(byte[] key, int prefixBytes) {
        try (ALock l = readLock.lock()) {
            for (Map.Entry<byte[], V> e : storage.entrySet()) {
                if (FastByteComparisons.compareTo(key, 0, prefixBytes, e.getKey(), 0, prefixBytes) ==
0) {
                    return e.getValue();
                }
            }

            return null;
        }
    }

    @Override
    public void updateBatch(Map<byte[], V> rows) {
        try (ALock l = writeLock.lock()) {
            for (Map.Entry<byte[], V> entry : rows.entrySet()) {
                put(entry.getKey(), entry.getValue());
            }
        }
    }

    public Map<byte[], V> getStorage() {
        return storage;
    }
}
```