java基础调查

都有其他语言基础

多数java 0基础，少数有基础

有java基础的学员，可先看看：

反射：https://liaoxuefeng.com/books/java/reflection/index.html
泛型：https://liaoxuefeng.com/books/java/generics/index.html

javase基础

参考地址：https://liaoxuefeng.com/books/java/introduction/index.html

快速过一遍

重点：OOP

第一个java代码

Hello.java

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

1. 手工编译运行

编译运行

```
javac Hello.java
java Hello
```

运行成功标志

输出Hello, world!

2. IDEA编译运行

1. 新建项目
2. 新建类（源码文件）
3. 修改类源码
4. 点击绿色播放键运行

java环境

- JDK：Java Development Kit
- JRE：Java Runtime Environment，java开发者工具包
- IDEA：开发工具

JDK8安装成功标志

打开cmd，执行

```
java -version
```

显示

> java version "1.8.0_401"
>
> Java(TM) SE Runtime Environment (build 1.8.0_401-b10)
>
> Java HotSpot(TM) 64-Bit Server VM (build 25.401-b10, mixed mode)

特点

以类为基本结构的语言

强类型

区分大小写

变量类型（基本常见类型）

- 整数类型：int，long
- 浮点数类型：float，double
- 字符类型：char
- 布尔类型：boolean，只有`true`和`false`两个值
- 定义一个字符串

```
String s = "hello";
// unicode字符
String s = "\u1111";
```

常量

```
final double PI = 3.14; // PI是一个常量
```

字符串

连接：+

引用类型的理解

```
// 字符串不可变
public class Main {
    public static void main(String[] args) {
        String s = "hello";
        String t = s;
        s = "world";
        System.out.println(t); // t是"hello"还是"world"?
    }
}
```

数组

1. 引用类型
2. 定义数组时长度要提前确定
3. 数组元素类型必须一致
4. 下标从0开始

```
// 数组
public class Main {
    public static void main(String[] args) {
        // 整形数组
        int[] ns = new int[5];
        ns[0] = 68;
        ns[1] = 79;
        ns[2] = 91;
        ns[3] = 85;
        ns[4] = 62;
        System.out.println(ns[0]);
        // 字符型数组
```

```
        String[] names = {
            "ABC", "XYZ", "zoo"
        };
        // 多维数组
        int[][] ns = {
            { 1, 2, 3, 4 },
            { 5, 6, 7, 8 },
            { 9, 10, 11, 12 }
        };
        System.out.println(ns.length); // 3
        System.out.println(ns[0][1]); // 2
    }
}
```

流程控制

和C++、PHP类似

异常处理

```
public class Main {
    public static void main(String[] args) {
        System.out.println("异常处理示例");
        int result = by(5, 0);
        System.out.println(result);
    }

    static int by(int a, int b) {
        try {
            // 用指定编码转换String为byte[]:
            return a / b;
        } catch (Exception e) {
            System.out.print("计算出错，错误信息：");
            System.out.println(e); // 打印异常信息
        }
        return 0;
    }
}
```

代码结构

src源码目录下通常会分门别类地存放很多源码，结构如下

模块-》包-》类（源码）

Java反序列化基础

要求：理解反序列化基础代码

1. 序列化和反序列化的基本代码

1.1 创建可序列化的 Java 类

首先，我们需要一个类，它实现 `Serializable` 接口，表示该类的对象可以被序列化和反序列化。

```java
import java.io.*;

// 实现 Serializable 接口的类
class Person implements Serializable {

    private String name;
    private int age;

    // 构造函数
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // 重写 toString 方法，方便输出信息
    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}
```

1.2 序列化过程

序列化是将对象转换为字节流，方便将字节流写入磁盘或发送到网络的过程。以下代码示例如何将 `Person` 对象序列化到文件中。

```java
import java.io.*;

public class SerializeExample {
    public static void main(String[] args) {
        Person person = new Person("Alice", 30);

        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("person.ser"))) {
            // 将 Person 对象写入到文件
            out.writeObject(person);
            System.out.println("对象已序列化");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

1.3 反序列化过程

反序列化是将字节流恢复为原始对象的过程。以下代码示例如何从文件中读取字节流并将其反序列化为 `Person` 对象。

```java
import java.io.*;

public class DeserializeExample {
    public static void main(String[] args) {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream("person.ser"))) {
            // 从文件中读取对象
            Person person = (Person) in.readObject();
            System.out.println("反序列化得到对象: " + person);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

1.4 如何运行

1. **编译代码**：

```
javac SerializeExample.java DeserializeExample.java
```

1. **运行序列化代码**：

```
java SerializeExample
```

这将会在当前目录生成一个名为 `person.ser` 的文件，其中保存了 `Person` 对象的序列化数据。

1. **运行反序列化代码**：

```
java DeserializeExample
```

程序将读取 `person.ser` 文件，并打印出反序列化后的 `Person` 对象。

1.5 输出结果

假设你先运行了 `SerializeExample` 类，然后运行 `DeserializeExample` 类，输出结果应为：

```
对象已序列化
反序列化得到对象: Person{name='Alice', age=30}
```

1.6 解释

1. **Serializable 接口**：`Person` 类实现了 `Serializable` 接口，表示这个类的对象可以被序列化。
2. **ObjectOutputStream**：将对象写入到字节流中（`person.ser` 文件）。
3. **ObjectInputStream**：从字节流中读取对象，并将其转换回原始对象。

通过这个简单的示例，您可以理解 Java 中如何使用序列化和反序列化来持久化对象或通过网络传输对象。

1.7 注意事项

- **serialVersionUID**：它是一个版本号，用于确保在反序列化时，类的版本是兼容的。如果序列化后的类版本与反序列化时的版本不一致，Java 会抛出 `InvalidClassException` 异常。
- **反序列化漏洞**：反序列化时，要小心接收来自不可信来源的数据，因为恶意数据可能会利用反序列化漏洞执行恶意代码。

Java反序列化漏洞基础

要求：能说出反序列化漏洞的原理。

原理：反序列化漏洞通常是指攻击者通过提供恶意构造的序列化数据，使得系统在反序列化时执行恶意代码或改变程序行为的漏洞。

Java 的反序列化漏洞常见于不安全地反序列化未经验证的数据。

在 Java 中，常使用 `ObjectInputStream` 类来进行反序列化操作。

示例代码

下面是一个简单的 Java 反序列化漏洞的示例 Demo。在这个例子中，我们：

1. 创建了一个类，其中包含一个简单的反序列化操作
2. 攻击者通过操控序列化数据来执行恶意代码。

```
import java.io.*;
import java.util.*;

public class DeserializationDemo {

    public static class VictimClass implements Serializable {
        private String name;

        public VictimClass(String name) {
            this.name = name;
        }

        public void sayHello() {
            System.out.println("Hello, " + name + "!");
        }
    }

    public static class ExploitClass implements Serializable {
        private String command;

        public ExploitClass(String command) {
            this.command = command;
        }

        // 重载 readObject 方法，在反序列化时执行恶意命令
        private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
            in.defaultReadObject();
            // 反序列化时执行恶意命令
            try {
                System.out.println("Executing exploit command: " + command);
```

```
                Runtime.getRuntime().exec(command); // 可能导致命令执行
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) throws Exception {
        // 创建正常的 VictimClass 实例
        VictimClass victim = new VictimClass("Victim");

        // 序列化 VictimClass 实例
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
        ObjectOutputStream objectOutputStream = new
ObjectOutputStream(byteArrayOutputStream);
        objectOutputStream.writeObject(victim);
        objectOutputStream.close();

        // 示例反序列化漏洞
        // 攻击者构造恶意 ExploitClass 实例
        ExploitClass exploit = new ExploitClass("notepad.exe"); // 这里使用的是
Windows 的例子
        ByteArrayOutputStream exploitStream = new ByteArrayOutputStream();
        ObjectOutputStream exploitObjectStream = new
ObjectOutputStream(exploitStream);
        exploitObjectStream.writeObject(exploit);
        exploitObjectStream.close();

        // 反序列化攻击者的恶意数据
        ByteArrayInputStream exploitInputStream = new
ByteArrayInputStream(exploitStream.toByteArray());
        ObjectInputStream exploitObjectInputStream = new
ObjectInputStream(exploitInputStream);
        exploitObjectInputStream.readObject(); // 执行恶意命令
        exploitObjectInputStream.close();
    }
}
```

2.2 示例说明

1. `VictimClass`：一个简单的可序列化类，包含一个 `sayHello` 方法。在实际场景中，这可能是目标对象。
2. `ExploitClass`：也是一个可序列化类，但它重载了 `readObject` 方法。在反序列化时，这个方法会执行恶意命令，利用 `Runtime.getRuntime().exec(command)` 来执行攻击者提供的命令。
3. **攻击过程**：
    - 反序列化恶意数据：攻击者通过构造一个带有恶意 `readObject` 方法的对象，并将其序列化。
    - 当反序列化这个恶意对象时，`readObject` 方法会执行攻击者指定的命令，在本例中是执行 `notepad.exe`。

2.3 运行示例

- 编译和运行该程序时，如果是 Windows 系统，攻击者构造的恶意命令将启动 `notepad.exe`。在 Linux 或 macOS 上，你可以使用 `ls`、`cat /etc/passwd` 等命令，具体取决于你的攻击目标。
- 你可以在 `ExploitClass` 中修改命令来执行任意操作，或删除 `System.out.println` 来隐藏恶意行为。

2.4 防御方法

为了防止 Java 反序列化漏洞，应该采取以下措施：

- **使用安全的反序列化库**：例如，使用像 `Jackson`、`Gson` 等库，它们允许你指定可接受的类。
- **避免直接反序列化未验证的用户数据**：永远不要直接反序列化来自不可信来源的数据。
- **禁用恶意类**：禁用 Java 默认反序列化过程中的危险类，如 `Runtime`、`ProcessBuilder` 等，可以通过配置类加载器或使用安全管理器来实现。
- **使用 `ObjectInputStream` 的 `resolveClass` 方法**：这可以帮助你限制反序列化时允许的类，避免加载不安全的类。

这个示例是一个典型的 Java 反序列化漏洞的示例。在生产环境中，你应该非常小心反序列化的使用，尤其是处理来自不受信任来源的数据时。