

# 《面向对象程序先导》

## Lec4-输入解析的结构化设计

北京航空航天大学计算机学院

吴际

2023.10.8

# 提纲

- 输入序列结构的特征分析
- 输入序列结构到对象
- 层次结构的映射
- 正则表达式的模式结构介绍
- 如何使用正则表达式
- 层次化方式来解析：层次正则
- 作业内容介绍

# 为什么需要输入解析

问题情境：写一个程序计算学生平均成绩，需要用户输入学生的姓名和某三门课的成绩。

如果用户输入的数据格式不正确或者解析出错，结果能正确吗？

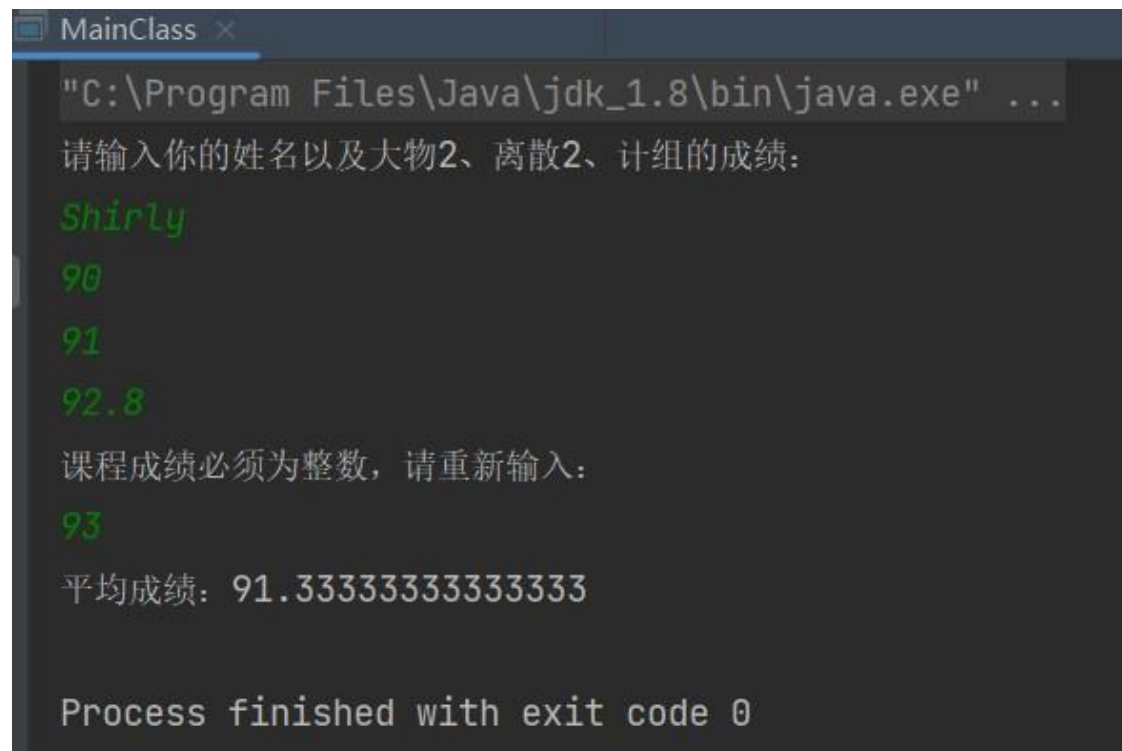
**Garbage in, garbage out.**

输入解析是确保程序准确性的第一步

# 为什么需要输入解析

输入解析：计算机程序对外部输入数据进行结构特征识别、验证和转换为程序内部数据结构的过程

- 输入结构的准确性与完整性
- 用户交互与体验
- 提高程序可维护性
  - 输入解析和业务处理逻辑分离



```
MainClass x
"C:\Program Files\Java\jdk_1.8\bin\java.exe" ...
请输入你的姓名以及大物2、离散2、计组的成绩：
Shirly
90
91
92.8
课程成绩必须为整数，请重新输入：
93
平均成绩： 91.33333333333333
Process finished with exit code 0
```

# 输入解析的步骤

## 1. 获取输入

## 2. 提取数据

- 分析输入序列结构的特征，提取需要的数据

## 3. 验证数据

- 验证输入数据的合法性，确保满足特定条件

## 4. 转换为程序内部数据

- 按照类数据结构的定义进行转换  
→ 构造对象

```
public class Student {  
    private String username;  
    private String birthday;  
    private String tel;  
  
    //构造方法：创建并初始化对象  
    public Student(String username, String birthday, String tel) {  
        this.username = username;  
        this.birthday = birthday;  
        this.tel = tel;  
    }  
}
```

# 输入序列结构的特征分析

- 输入数据的元素类型
  - 字符串
    - 整数、浮点数、日期、字符串等
- 输入数据元素之间的关系
  - 格式（语法）上有序
  - 实际含义可以是无序
- 输入数据序列的长度
  - 字符长度
  - 有独立含义数据项的个数

每行对应结构和含义明确的操作：{adv\_id}是数字元素，{name}是字符串元素。。。

type	attribute	意义
1	{adv_id} {name}	加入一个 ID 为 {adv_id}、名字为 {name} 的冒险者
2	{adv_id} {bot_id} {name} {capacity}	给 ID 为 {adv_id} 的冒险者增加一个药水瓶，药水瓶的 ID、名字、容量分别为 {bot_id}、{name}、{capacity}，且默认为已装满 (isEmpty == false)
3	{adv_id} {bot_id}	将 ID 为 {adv_id} 的冒险者的 id 为 {bot_id} 的药水瓶删除
4	{adv_id} {equ_id} {name} {star}	给 ID 为 {adv_id} 的冒险者增加一个装备，装备的 ID、名字、星级分别为 {equ_id}、{name}、{star}

有序

```
6
1 1 a1
1 2 a2
4 1 11 b1 5
2 2 21 c1 100
6 1 11
9 1 11
```

由操作类型type可知该操作内包含的元素个数

# 输入序列结构的特征分析

type	attribute	
1	{adv_id}	6
	{name}	1 1 a1
2	{adv_id}	1 2 a2
	{bot_id}	
	{name}	4 1 11 b1 5
	{capacity}	2 2 21 c1 100
3	{adv_id}	6 1 11
	{bot_id}	9 1 11
4	{adv_id} {equ_id} {name} {star}	给 ID 为 {adv_id} 的冒险者增加一个装备, 装备的 ID、名字、星级分别为 {equ_id}、{name}、{star}

- 序列分隔符
  - 空格、换行、逗号.....
- 序列的结构
  - 重复模式结构：提取和处理相似数据
  - 层次嵌套：如树状结构、多层次数据

# 输入序列结构的特征分析

- 课程平台用户信息结构
  - 序列结构+嵌套结构：
    - 用户信息：用户名，生日，手机号，学习的课程（多个）
    - 课程信息：课程id、课程名和成绩
- 需要能够识别序列结构+嵌套结构
- 然后逐个逐层解析和提取出用户信息和课程信息
- 最后再按照信息的嵌套层次关系构造相应的对象

元素类型：结构化用户信息

序列结构：嵌套结构，每个用户基本信息对象包含了一个嵌套的课程列表

序列分隔符：逗号

元素之间的关系：  
数组中对象无序

```
[
  {
    "username": "Alice",
    "birthday": "2001-08-02",
    "tel": "13928749033",
    "courses": [
      {
        "courseId": "CSCI101",
        "courseName": "Introduction to Computer Science",
        "grade": 92
      },
      {
        "courseId": "MATH201",
        "courseName": "Advanced Mathematics",
        "grade": 85
      }
    ]
  },
  {
    "username": "Bob",
    "birthday": "2006-02-08",
    "tel": "17642563841",
    "courses": [
      {
        "courseId": "ENG101",
        "courseName": "English Composition",
        "grade": 88
      },
      {
        "courseId": "HIST202",
        "courseName": "World History",
        "grade": 78
      }
    ]
  }
]
```



# 输入序列结构到对象层次结构的映射

- 用于将字符串的输入序列转换为具有层次化结构的对象表示
- 输入解析
  - 将输入数据从其原始格式（如文本、JSON、XML等）转换为程序所能计算处理的数据（结构）的过程
- 对象构造
  - 将输入解析后的数据（结构）映射到对象（层次结构）的过程
  - 考虑输入序列结构和业务需求，设计合适的对象模型（数据结构设计）
    - 提供合适的构造方法来创建对象并组装成合适的层次结构

# 输入序列结构到对象层次结构的映射

简化用户信息列表结构：

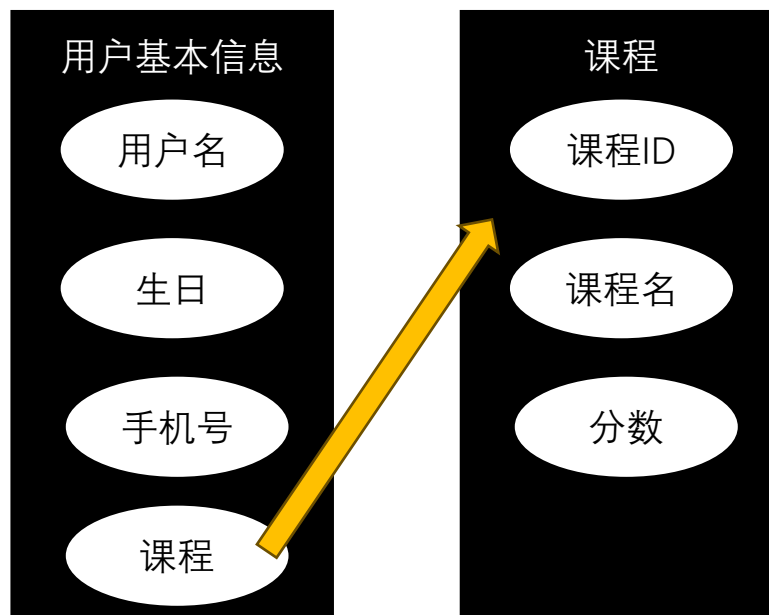
**第一行是个整数，表示用户数量**

接下来对于每个用户基本信息，  
第一行为用户名、用户生日、用户手机号码，用空格隔开

**第二行是个整数，学习的课程数**

接下来若干行，每行输入一门课程信息：课程ID，课程名，用户取得的分数，用逗号隔开

```
1
Alice 2001-08-02 13928749033
2
CSCI101,Introduction to Computer Science,92
MATH201,Advanced Mathematics,85
```



设计实现对象层次结构

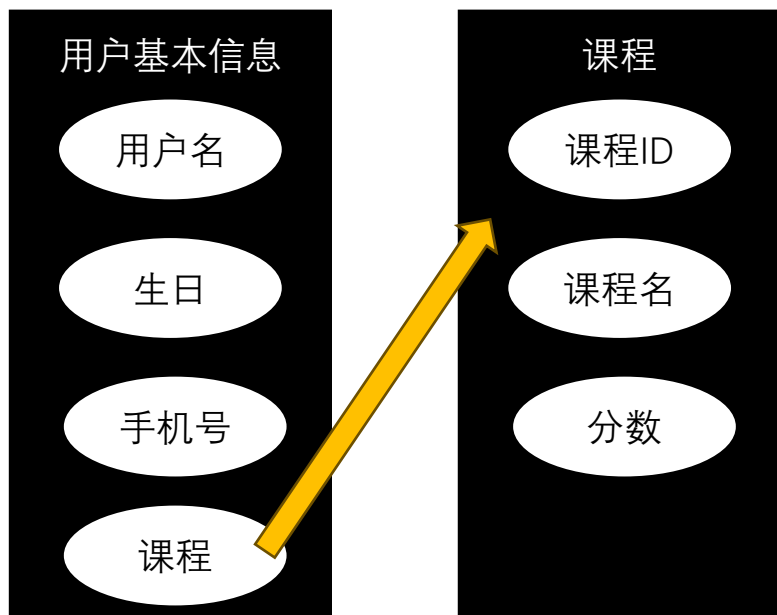
```
public class Student {
    private String username;
    private String birthday;
    private String tel;
    private ArrayList<Course> courses;
    public Student() {
        courses = new ArrayList<>();
    }

    // Getter and setter methods
}

public class Course {
    private String courseId;
    private String courseName;
    private int grade;

    // Getter and setter methods
}
```

# 输入序列结构到对象层次结构的映射



```
1
Alice 2001-08-02 13928749033
2
CSCI101,Introduction to Computer Science,92
MATH201,Advanced Mathematics,85
```

将给定的文本输入映射到对象层次结构

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<Student> students = new ArrayList<>();
    while (scanner.hasNext()) { // 创建学生实例
        Student student = new Student();
        student.setUsername(scanner.next());
        student.setBirthday(scanner.next());
        student.setTel(scanner.next());
        int numCourses = scanner.nextInt();
        for (int i = 0; i < numCourses; i++) { // 创建课程实例
            Course course = new Course();
            course.setCourseId(scanner.next());
            course.setCourseName(scanner.next());
            course.setGrade(scanner.nextInt());
            student.addCourse(course); // 添加到学生学习课程数组内
        }
        students.add(student);
    }
    scanner.close();
}
```

# 使用String来解析输入序列

- split 基于**正则表达式**切分字符串
  - split(String regex)
  - split(String regex, int limit):限制返回字符串的个数

```
student.setUsername(scanner.next());  
student.setBirthday(scanner.next());  
student.setTel(scanner.next());
```

```
String line = scanner.nextLine();  
/* 使用空格进行字符串分割 */  
String[] group = line.split(" ");  
student.setUsername(group[0]);  
student.setBirthday(group[1]);  
student.setTel(group[2]);
```

- substring
  - 返回字符串的子串
- indexOf
  - 返回字符串中指定字符第一次出现的位置
- trim
  - 删除字符串两侧的空格

**正则表达式：**使用一种带有特殊符号的字符串来定义一个普通字符串的序列结构特征

# 验证输入结构的合法性

提取数据后，需要验证用户基本信息输入中的生日和手机号码是否合法。生日需要满足YYYY-MM-DD格式(Y,M,D必须为数字，暂不考虑月、日范围要求)，手机号码要求为11位数字。

```
public boolean isValidDate(String date) {  
    /* 日期位数为10 */  
    if (date.length() != 10)  
        return false;  
    /* split分割年月日 */  
    String[] groups = date.split("-");  
    if (groups.length != 3)  
        return false;  
    /* 判断年月日是否为数字 */  
    for (int i = 0; i < 3; i++)  
        for (int j = 0; j < groups[i].length(); j++)  
            if (!Character.isDigit(groups[i].charAt(j)))  
                return false;  
    return true;  
}
```

循环判断

```
public boolean isValidTel(String tel) {  
    int len = tel.length();  
    if (len != 11) System.out.println("error");  
    for (int i = 0; i < len; i++) {  
        char ch = tel.charAt(i);  
        if (ch < '0' || ch > '9')  
            return false;  
    }  
    return true;  
}
```

为每一种判断逻辑编写代码实在是太繁琐了。有没有更简单的方法？

# 正则表达式

使用正则表达式来准确定义待处理字符串的序列结构特征（如手机号11个字符全是数字）

String类提供了基于正则表达式的匹配方法，快速判断一个字符串的结构是否与给定正则表达式所定义的结构一致

```
public boolean isValidTel(String tel) {  
    // ...  
    return tel.matches("\\d{11}");  
    // ...  
}
```

正则表达式帮助我们避免编写大量的手动验证代码，而是通过一种更简洁的方式来实现输入的验证和解析。

# 正则表达式

- 模式匹配
  - 查找和识别文本中特定格式的数据
- 搜索、提取数据
  - 从字符串中提取特定信息
- 输入验证
  - 验证输入数据是否符合特定格式要求
- 语法分析
  - 编译器、解析器等领域，用于对代码或文本进行语法分析和处理
- 表单验证
  - Web开发，用于验证表单数据的合法性和完整性

# 如何使用正则表达式

- 模糊匹配

符号	意义	示例
.	匹配任意单个字符	ab.:可以匹配abc,ab1,ab/, 但不能匹配ab,abcc
\d	匹配一个数字	1\d1:可以匹配101,111, 但不能匹配11,1001
\w	匹配一个字母、数字或下划线	OO\w:可以匹配OO_,OOQ,OO7, 但不能匹配OO?
\D	匹配非数字	
\s	匹配一个空格字符（空格和tab字符）	
\n	匹配一个换行符	
\r	匹配一个回车符	



# 如何使用正则表达式

- 重复匹配：如何匹配多个数字？
  - \* 可以匹配0或多个字符，如 `Object\d*`,可匹配`Object1`,`Object123...`
  - + 可以匹配至少一个字符
  - ? 可以匹配0或1个字符
  - {} 指定重复次数，如`A\d{3}`

```
/* 正则表达式写法 */  
public boolean isValidDate(String date) {  
    return date.matches("\\d{4}-\\d{2}-\\d{2}");  
}
```

# 如何使用正则表达式

- 复杂匹配规则

- 匹配开头和结尾, ^表示开头, \$表示结尾
  - ^1\d{2}, \d{2}1\$
- 匹配指定范围
  - []表示匹配范围内的字符, 如[3-9]可以匹配3~9
  - [^A-F]表示匹配A~F之外的任意字符
- 或规则匹配
  - apple|banana 表示可以匹配apple或banana

```
tel.matches("^1[3-9]\\d{9}");
```

除了是11位数字外, 手机号码必须以1开头, 第二位数字只能是3~9.

```
pwd.matches("\\w*[^a-z0-9]+\\w*");
```

# 如何使用正则表达式

匹配成功后想提取年月日分别存储，那么如何提取匹配的子串？

1.String的substring()和indexOf()

2.分组匹配

分组匹配（需引入java.util.regex包）

1. 用()把要提取的规则分组：

"(\\d{4})-(\\d{2})-(\\d{2})"

2. 按括号提取子串：group(1)...

group(0)提取输入的整个字符串

```
public boolean isValidDate(String date) {  
    return date.matches("\\d{4}-\\d{2}-\\d{2}");  
}
```

```
public Birthday getDate(String date) {  
    Birthday birthday = new Birthday(); // Birthday 对象，存储日期信息  
    /* 使用Pattern类编译正则表达式 */  
    Pattern pattern = Pattern.compile("(\\d{4})-(\\d{2})-(\\d{2})");  
    /* 使用Pattern对象匹配date */  
    Matcher matcher = pattern.matcher(date);  
    if (matcher.matches()) {  
        // 匹配成功，提取年、月、日并存储到 Birthday 对象中  
        birthday.setYear(matcher.group(1));  
        birthday.setMonth(matcher.group(2));  
        birthday.setDay(matcher.group(3));  
        return birthday;  
    } else {  
        // 不匹配，进行必要的错误处理  
    }  
}
```

# 如何使用正则表达式

## • 非贪婪匹配

有一个字符串，形如 “ayjuhggoodgoodgood”，将其切分成两个子串，第二个子串为该字符串末尾的连续的good，另一个子串为字符串剩余部分

`\w+` 可以匹配后面任意个good

```
public static void main(String[] args) {  
    Pattern pattern = Pattern.compile("(\\w+?)((good)*")  
    Matcher matcher = pattern.matcher("ayjuhggoodgoodgood");  
    if (matcher.matches()) {  
        System.out.println("group1=" + matcher.group(1));  
        System.out.println("group2=" + matcher.group(2));  
    }  
}
```

```
"C:\Program Files\Java\jdk_1.8\bin\java.exe" ...  
group1=ayjuhggoodgoodgood  
group2=  
  
Process finished with exit code 0
```

在规则`\w+`后面加个?即可表示非贪婪匹配

正则表达式默认使用贪婪匹配：任何一个规则，它总是尽可能多地向后匹配。

# 如何使用正则表达式

- 基于正则表达式的匹配提取是一种基于模式匹配的方式，经常需要考虑整个输入的上下文信息来进行处理
- 上下文：待提取信息的前序或后序字符串的特征
  - 在模式设计时需要考虑该模式前后的文本内容以及相对位置关系
- 经典场景：特定序列的符号匹配
  - 左右括号匹配：多种括号
  - 相对顺序、嵌套关系以及中间可能出现的其他字符

# 如何使用正则表达式

一行文本包含数条用户基本信息

每条用户信息最后都有逗号

**找出学习了课程名包含  
“Computer” 的用户**

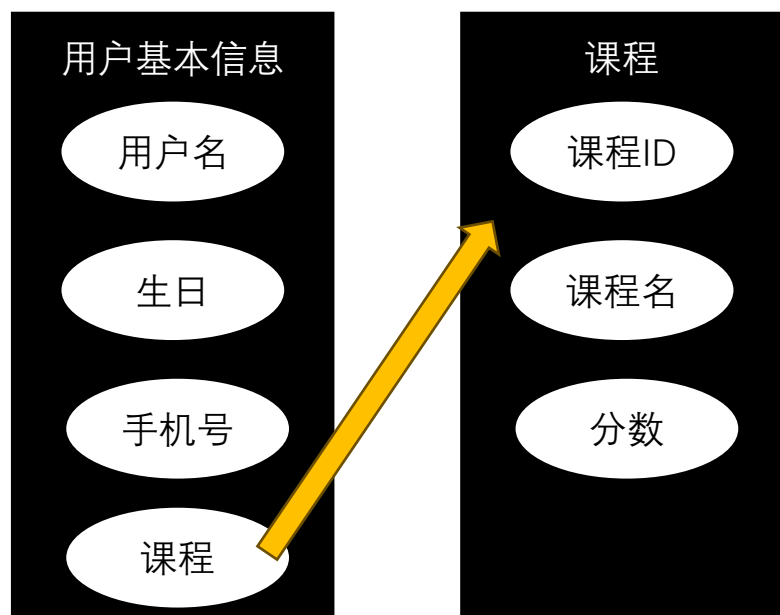
```
{"username": "Alice", "birthday": "2001-08-02",  
"tel": "13928749033", "courses": [{"courseId": "CSCI101",  
"courseName": "Introduction to Computer Science", "grade": 92},  
{"courseId": "MATH201", "courseName": "Advanced Mathematics",  
"grade": 85}]}, {"username": "Bob", "birthday": "2006-02-08",  
"tel": "17642563841", "courses": [{"courseId": "ENG101",  
"courseName": "English Composition", "grade": 88}, {"courseId":  
"HIST202", "courseName": "World History", "grade": 78}]},
```

从序列结构中寻找层次

- 每条用户基本信息是一个JSON对象，以嵌套结构来包含属性 `courses`，包含多个课程信息的JSON对象
- 先匹配用户基本信息，再提取课程信息数组，从中寻找 `courseName` 含有 “Computer” 的课程

# 层次化解析：层次正则

- 将复杂的输入序列结构分解成多个层次的正则表达式模式，以便逐层匹配和提取所需的信息
- 涉及多层嵌套、多个子元素



```
{"username": "Alice", "birthday": "2001-08-02",  
"tel": "13928749033", "courses": [{"courseId": "CSCI101",  
"courseName": "Introduction to Computer Science", "grade": 92},  
{"courseId": "MATH201", "courseName": "Advanced Mathematics",  
"grade": 85}]}, {"username": "Bob", "birthday": "2006-02-08",  
"tel": "17642563841", "courses": [{"courseId": "ENG101",  
"courseName": "English Composition", "grade": 88}, {"courseId":  
"HIST202", "courseName": "World History", "grade": 78}]}
```

# 层次化解析：层次正则

为每个层次构建正则模式

分析输入序列结构特征，找到用户基本信息的特征模式

- 每个JSON对象包含在 “{}” 中，用户基本信息、属性之间用 “,” 分隔
- 直接用 “(\\{.\*?\\})” 提取用户基本信息如何？

“courses”中的每条课程信息都包含在“{}”里，“(\\{.\*?\\})”可以匹配到课程信息“{}”

“}” 是用户基本信息的结束标志

```
{ "username": "Alice", "birthday": "2001-08-02",  
  "tel": "13928749033", "courses": [ { "courseId": "CSCI101",  
    "courseName": "Introduction to Computer Science", "grade":  
    92 }, { "courseId": "MATH201", "courseName": "Advanced  
Mathematics", "grade": 85 } ], { "username": "Bob", "birthday":  
  "2006-02-08", "tel": "17642563841", "courses": [ { "courseId":  
    "ENG101", "courseName": "English Composition", "grade":  
    88 }, { "courseId": "HIST202", "courseName": "World History",  
    "grade": 78 } ] },
```

“\\{”是转义表达，表示匹配‘{’  
需要转义的字符 '\$', '(', ')', '\*', '+', '.', '[', ']', '?', '\\', '^', '{', '}', '|’



# 层次化解析：层次正则

为每个层次构建模式

## (1) 匹配用户基本信息

```
public static void hierarchicalRegex(String line) {  
    /* 正则表达式切分用户基本信息 */  
    Pattern pt1 = Pattern.compile("(\\{.*?\\})", "");  
    Matcher mt1 = pt1.matcher(line);  
    for (int i = 0; i < mt1.groupCount(); i++) {  
        String userInfo = mt1.group(i+1);  
        // ...  
    }  
}
```

```
{"username": "Alice", "birthday": "2001-08-02",  
"tel": "13928749033", "courses": [{"courseId": "CSCI101",  
"courseName": "Introduction to Computer Science",  
"grade": 92}, {"courseId": "MATH201", "courseName":  
"Advanced Mathematics", "grade": 85}]}, {"username":  
"Bob", "birthday": "2006-02-08", "tel": "17642563841",  
"courses": [{"courseId": "ENG101", "courseName":  
"English Composition", "grade": 88}, {"courseId":  
"HIST202", "courseName": "World History", "grade":  
78}]},
```

# 层次化解析：层次正则

为每个层次构建模式

(2) 提取 courses 的值

分析输入序列结构特征

- courses 的值是数组 [ ]
- 正则表达式：  
(**"courses"**:**(\\[.\*?\\])**)

```
{username: "Bob", "birthday": "2006-02-08",  
  "tel": "17642563841", "courses": [{"courseId":  
    "ENG101", "courseName": "English Composition",  
    "grade": 88}, {"courseId": "HIST202", "courseName":  
    "World History", "grade": 78}]}
```

# 层次化解析：层次正则

为每个层次构建模式

(2) 提取 courses 的值

find依次向后匹配，如果匹配成功，返回true，否则false；通过group(1)取到匹配的结果

```
public static void hierarchicalRegex(String line) {  
    /* 正则表达式切分用户基本信息 */  
    Pattern pt1 = Pattern.compile("(\\{.*?\\}\\})",");  
    Matcher mt1 = pt1.matcher(line);  
    while (mt1.find()) {  
        String userInfo = mt1.group(1);  
        /* 提取courses的值 */  
        Pattern pt2 = Pattern.compile("\"courses\":(\\[.*?\\])");  
        Matcher mt2 = pt2.matcher(userInfo);  
        if (mt2.find()) {  
            String courses = mt2.group(1);  
            // ...  
        } else {  
            // error  
        }  
    }  
}
```

# 层次化方法解析：层次正则

为每个层次构建模式

(2) 提取 courses 中的  
每门课程的courseName

```
[{"courseId": "ENG101", "courseName": "English  
Composition", "grade": 88}, {"courseId":  
"HIST202", "courseName": "World History",  
"grade": 78}]
```

*/\* 提取courses的值 \*/*

```
Pattern pt2 = Pattern.compile("\\courses\\":(\\[.*?\\]));  
Matcher mt2 = pt2.matcher(line);
```

```
if (mt2.find()) {
```

```
    String courses = mt2.group(1);
```

*/\* 判断用户是否学习了课程名包含Computer的课程 \*/*

```
    boolean flag = false;
```

```
    Pattern pt3 = Pattern.compile("\\courseName\\": \\(.*?\\)");
```

```
    Matcher mt3 = pt3.matcher(courses);
```

```
    while (mt3.find()) {
```

```
        String courseName = mt3.group(1);
```

```
        if (courseName.contains("Computer")) {
```

```
            flag = true;
```

```
        }
```

```
    }
```

```
    if (flag) {
```

```
        System.out.println(userInfo);
```

```
    }
```

```
}
```

# 作业内容介绍

- 引入战斗模式和战斗日志
  - 输入解析
    - 正则表达式解析
    - 非法输入判断
- 战斗模式中冒险者相关属性的增减
- 战斗模式与战斗日志的记录与查询功能
  - 如何存储战斗日志记录方便查询?
  - 如何筛选符合条件的记录?