# CommandParser — Java 命令行解析器

作者: 张柳彬

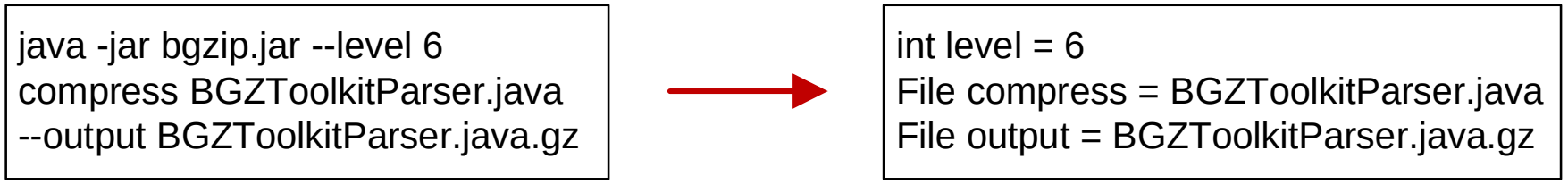
Precision Medical Genomics Laboratory (PMGLab)

Sun Yat-sen University
ZhongShan School of Medicine

# 简介

**命令行界面 (Command-line interface, CLI)** 是一种基于文本的用户界面，用于运行程序、管理计算机文件以及与计算机交互。命令行界面与软件图形界面、Web 服务一样，都是用于实现程序操作的内部形式与人类可以接受的形式之间的转换。通常，命令行界面接受用户键盘输入的指令，并将指令解析为不同的程序运行时参数或设置，最终发起后端的计算任务。

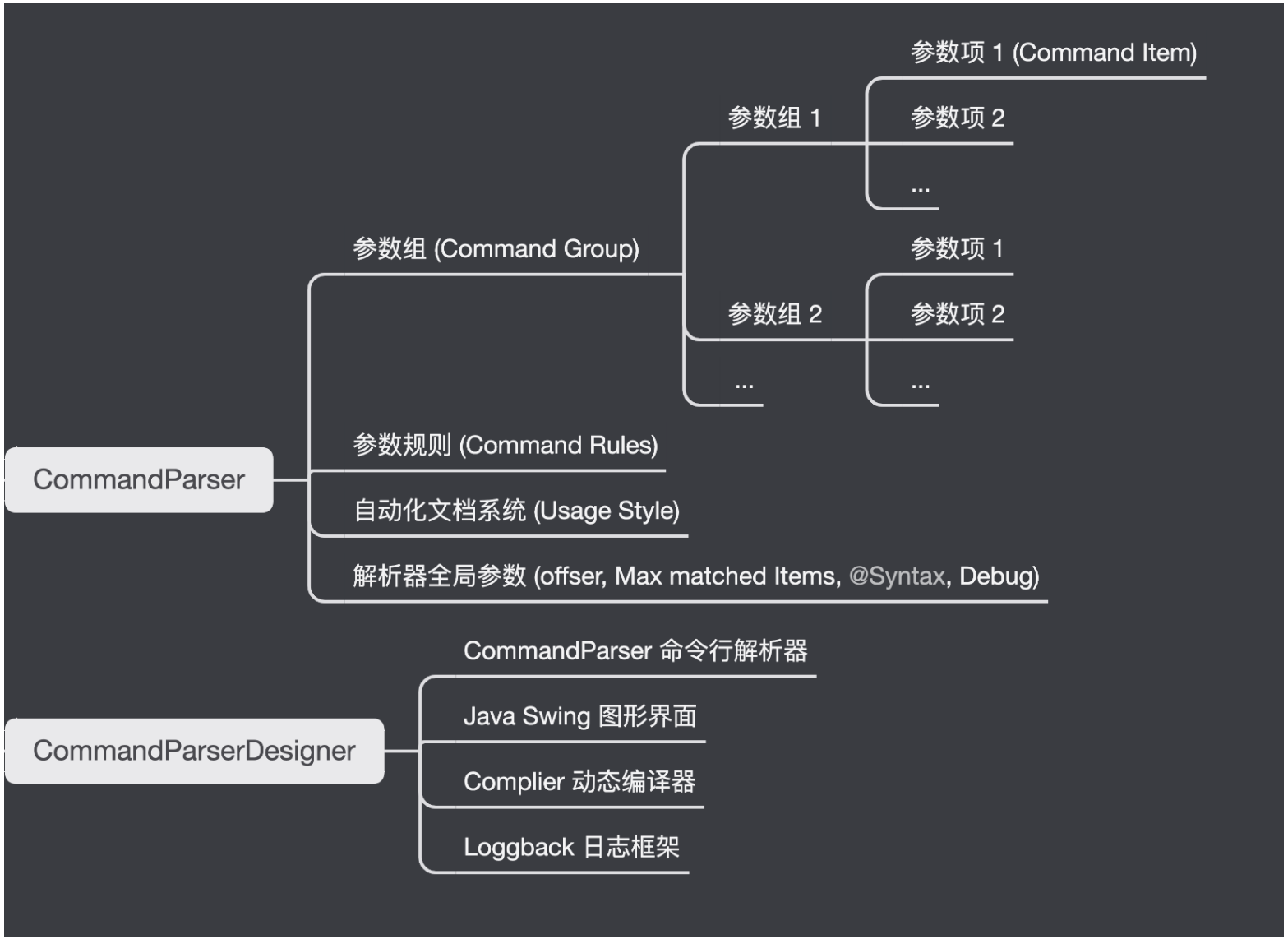
在 Java 中将程序封装为可运行的 jar 包，能够方便不同平台之间传输、使用，而不需要考虑 IDE 环境而进行繁琐的配置。通常 jar 包就包含了完成一套计算/分析/服务所需的全部代码，为了让 jar 包可以根据不同的输入参数执行不同的任务，以提升开发效率、用户交互体验，就需要进行命令行开发。




用户输入　　参数解析

CommandParser 是一个基于 Java 平台开发的轻量级框架，用于快速地开发、解析、管理命令行参数。它提供了一个基本的图形界面 (CommandParserDesigner)，用于可视化地管理、编辑命令项目。

# CommandParser 框架



# 下载与安装 CommandParser

CommandParser 在 JDK 8 中开发完成, 得益于 Java 跨平台及向下兼容的特性, 它也可以在所有支持 Java 语言的软件与硬件环境中运行.

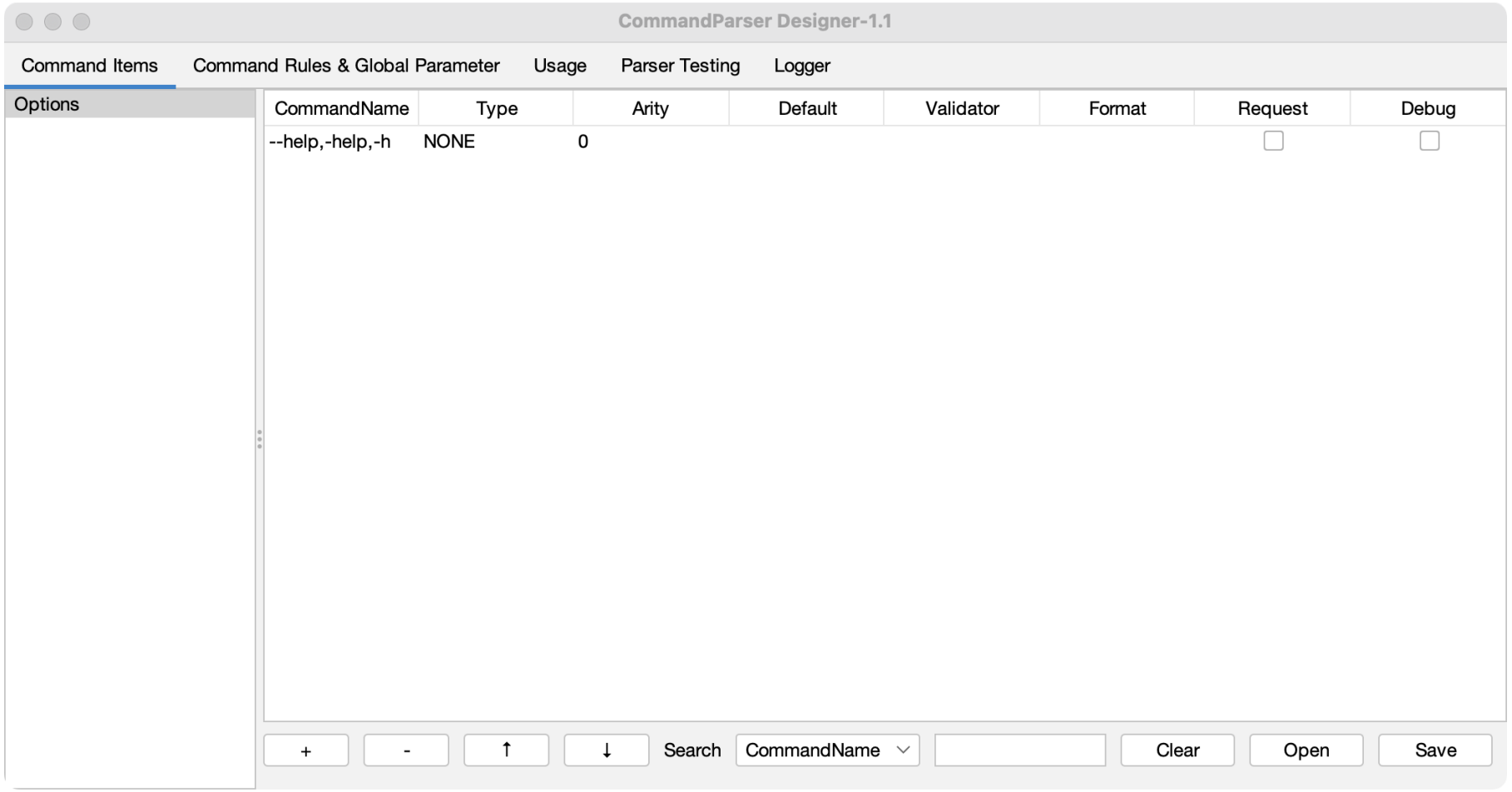
commandParser-1.1.jar: 用于导入其他项目中构建最小运行环境;

commandParserDesigner-1.1.jar: 用于启动图形界面进行管理与开发.

| 资源类型 | 文件 |
| --- | --- |
| 软件包 | http://pmglab.top/commandParser/commandParser-1.1.jar |
| 设计器图形界面 | http://pmglab.top/commandParser/commandParserDesigner-1.1.jar |
| 源代码 | https://github.com/Zhangliubin/commandParser-1.1 |
| 说明文档 | http://pmglab.top/commandParser/ |
| API 文档 | http://pmglab.top/commandParser/api-docs/ |
| 示例文件 | http://pmglab.top/commandParser/BGZToolkitParser.java |

# 启动设计器图形界面

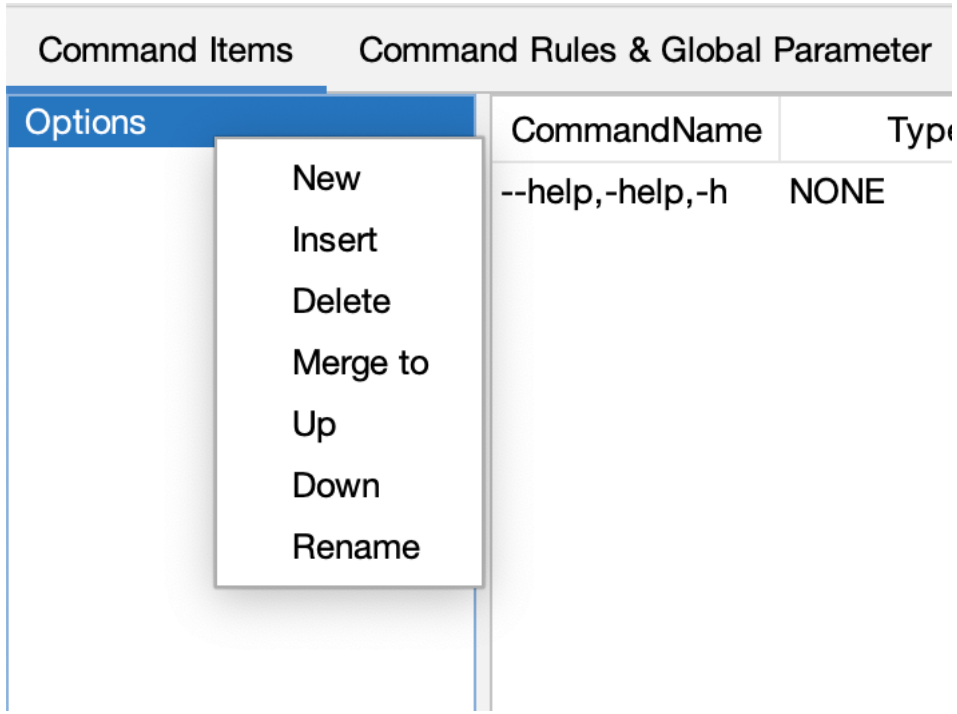
在终端中输入 java -jar ./commandParserDesigner-1.1.jar 指令或双击软件, 启动图形界面.



# 管理参数组 (Command Group)

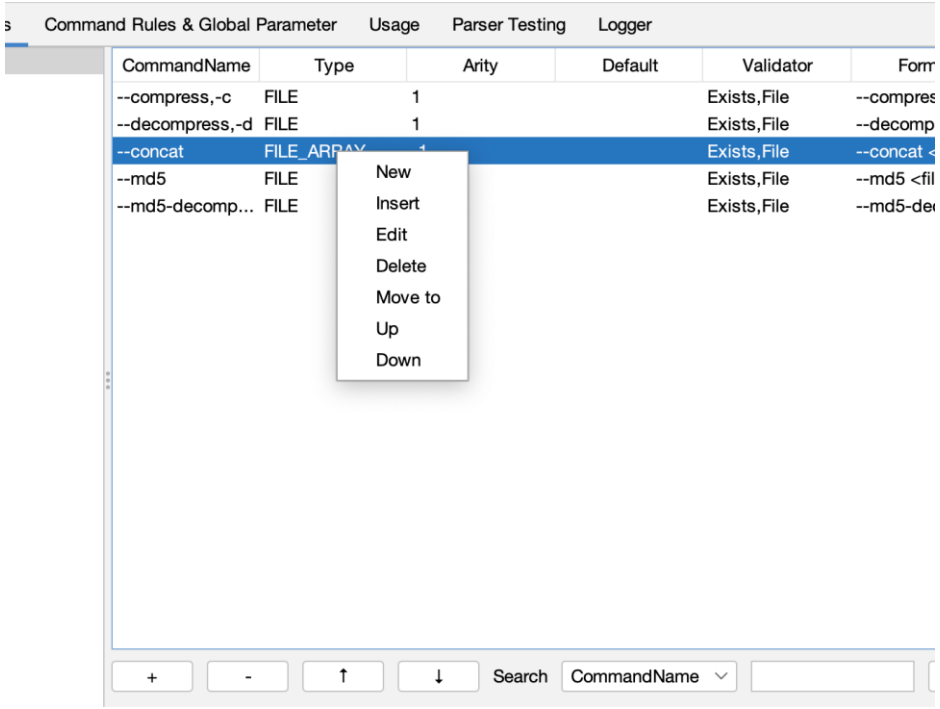
在 Command Items 标签页中, 左侧为参数组面板. 在参数组面板空白处或选中参数组点击鼠标右键, 展开管理菜单. 管理菜单包含以下 7 种操作:

- New: 创建新参数组
- Insert: 在当前位置插入新参数组
- Delete: 删除该参数组
- Merge to: 将该参数组合并至其他参数组
- Up: 上移该参数组
- Down: 下移该参数组
- Rename: 重命名参数组



# 管理参数项 (Command Items)

在 Command Items 标签页中, 右侧为参数项面板. 在参数项面板空白处或选中参数项点击鼠标右键, 展开管理菜单. 管理菜单包含以下 7 种操作:

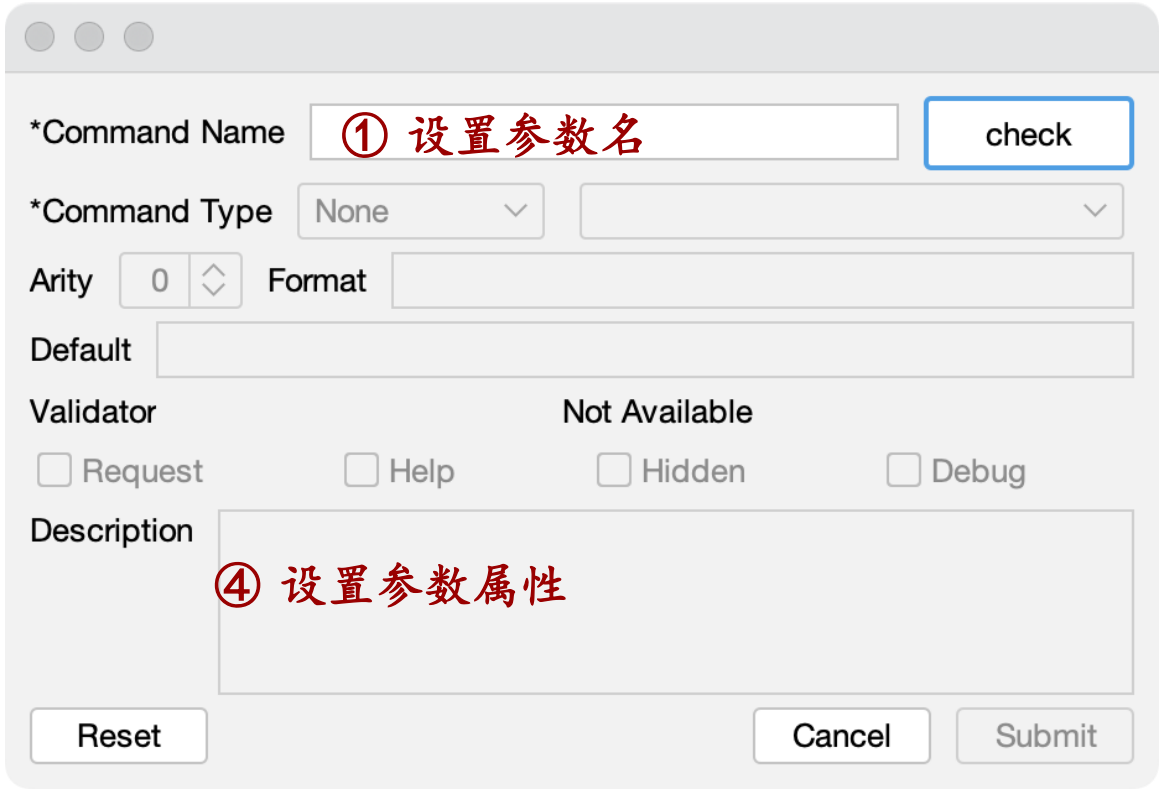



- **New:** 创建新参数项 (快捷键: Ctrl + N)
- **Insert:** 在当前位置插入新参数项
- **Edit:** 编辑该参数项 (快捷键: 双击参数项)
- **Delete:** 删除该参数项 (快捷键: Ctrl + Delete)
- **Merge to:** 将该参数项移动至其他参数组
- **Up:** 上移该参数项 (快捷键: Ctrl + U)
- **Down:** 下移该参数项 (快捷键: Ctrl + D)

下方菜单栏 "+" 对应 New 操作, "-" 对应 Delete 操作, "↑" 对应 Up 操作, "↓" 对应 Down 操作

# 编辑参数项

新建或编辑参数项时，弹出参数项子面板，子面板有 12 个参数属性.




② 点击进行参数名规范检查


③-1 通过，此时解锁其他控件

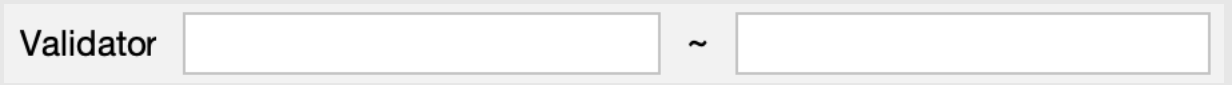
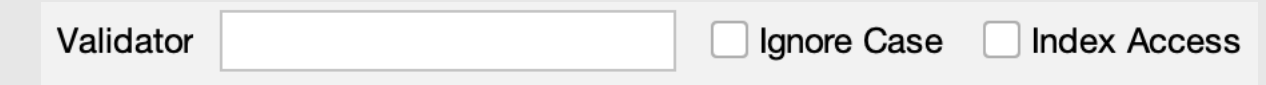
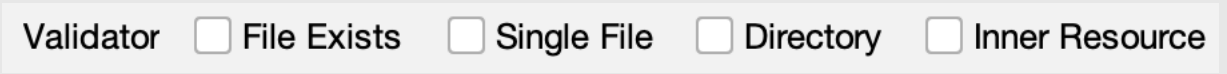

③-2 失败，参数名重复或包含非法字符

⑤ 提交参数项

# 编辑参数项

| 参数属性 | 描述 |
| --- | --- |
| **Command Name**<br>参数名 | 1. 参数具有多个参数名时，使用逗号分隔 (如: --output,-o)，第一个参数名作为主参数名<br>2. 参数名格式: 0-9a-zA-Z+-_<br>3. 参数名输入完成后点击 check 按钮进行检查 (检查是否符合格式、是否重名) |
| **Command Type**<br>参数类型 | 1. 内置 None (无类型), Boolean, Byte, Short, Integer, Long, Float, Double, String, File 10 种基本类型<br>2. 数据类型派生 Value, Array, Set, Range 等类型 |
| **Validator**<br>参数验证器 | 不同参数类型可用的验证器种类不同 (文件验证器、数值验证器等)，详见 不同参数类型支持的验证器 |
| **Default**<br>默认值 | 该默认值需要经过 Command Type 进行格式转换及 Validator 参数值验证<br>输入格式与 Format 和 Arity 定义的格式一致 |
| **Arity** 参数长度 (-1 代表不定长); **Format** 输入格式提示, 默认根据参数类型设置; **Description** 参数描述 | |
| **Request** 是否为必备参数; **Help** 是否识别为帮助指令; **Hidden** 是否在文档中隐藏该指令; **Debug** 调试指令 | |

# 不同参数类型 (Command Type) 支持的验证器 (Validator)

| Command Type | Validator 支持的类型 |
| --- | --- |
| None, Boolean | 不支持使用验证器 |
| Byte, Short, Integer, Long, Float, Double | 支持两种数值范围验证器:<br>1. 范围: 最小值 ~ 最大值<br>2. 指定最小值: ≥ 最小值<br>  |
| String | 限定值验证器<br>多个限定值使用空格分隔<br>Ignore Case 忽略大小写, Index Access 允许使用索引访问 (0 代表第一个限定值…)<br>  |
| File | 文件验证器<br>File Exists 文件路径必须存在<br>Single File 文件路径不能指向文件夹<br>Directory 文件路径必须指向文件夹<br>Inner Resource 优先识别当前运行环境资源 (允许访问 jar 包内部文件)<br>  |

# 参数类型及对应的默认格式

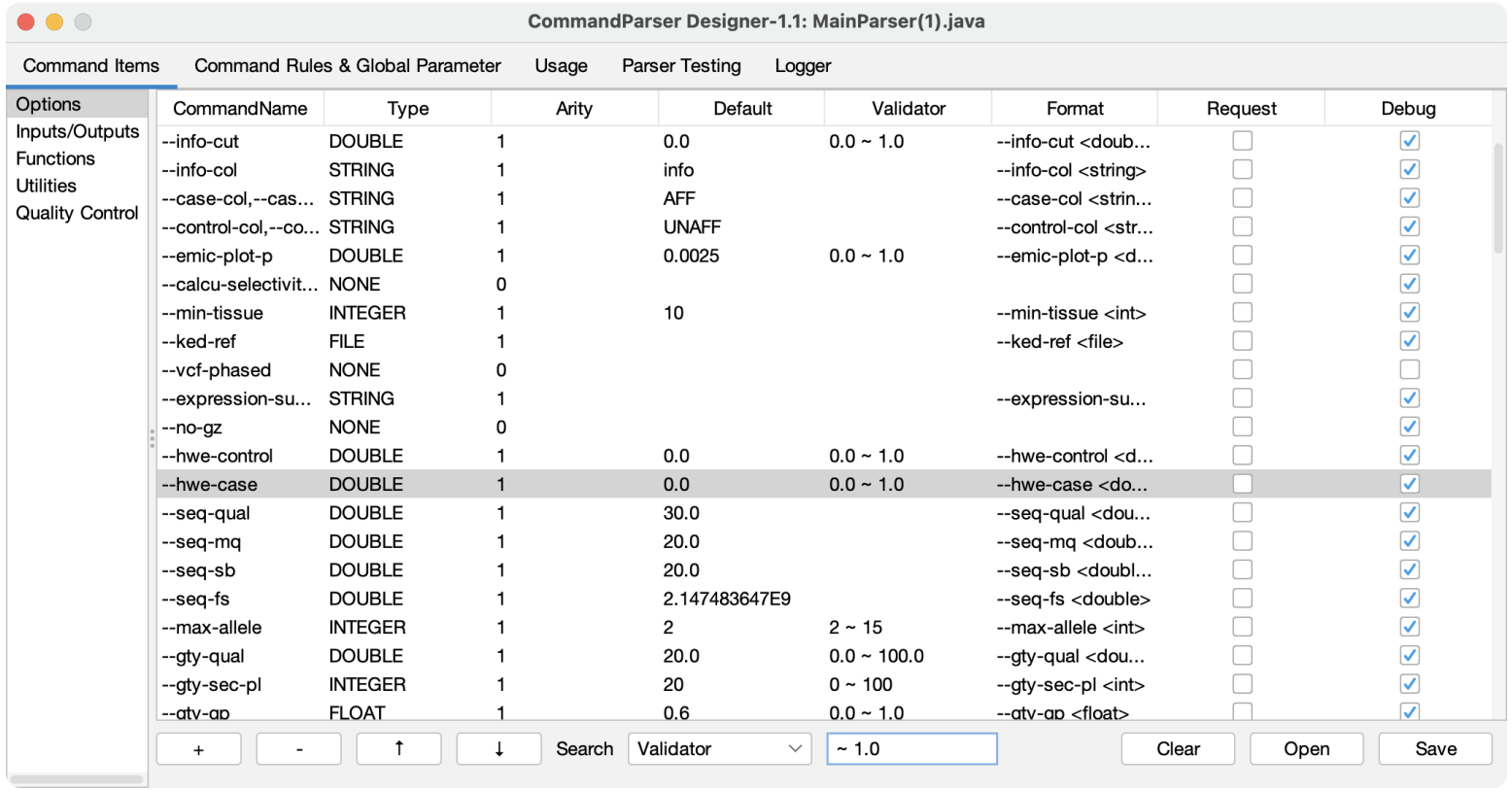
| Command Type | 默认格式 (Format) |
|---|---|
| VALUE | value |
| ARRAY | value value … |
| ARRAY_COMMA | value,value,… |
| ARRAY_SEMICOLON | value;value;… |
| SET | value value … |
| SET_COMMA | value,value,… |
| SET_SEMICOLON | value;value;… |
| MAP | key=value key=value … |
| MAP_COMMA | key=value,key=value,… |
| MAP_SEMICOLON | key=value;key=value;… |

| Command Type | 默认格式 (Format) |
|---|---|
| RANGE | value-value |
| LABEL_RANGE | label:value-value |
| LABEL_RANGE_COMMA | label:value-value,label:value-value,… |
| LABEL_RANGE_SEMICOLON | label:value-value;label:value-value;… |
| LABEL_ARRAY | label:value,value,… label:value,value,… |
| LABEL_ARRAY_SEMICOLON | label:value,value,…;label:value,value,…;… |

key, label 对应的 Java 数据类型为 String
value 类型与相应的参数类型有关

# 搜索参数项

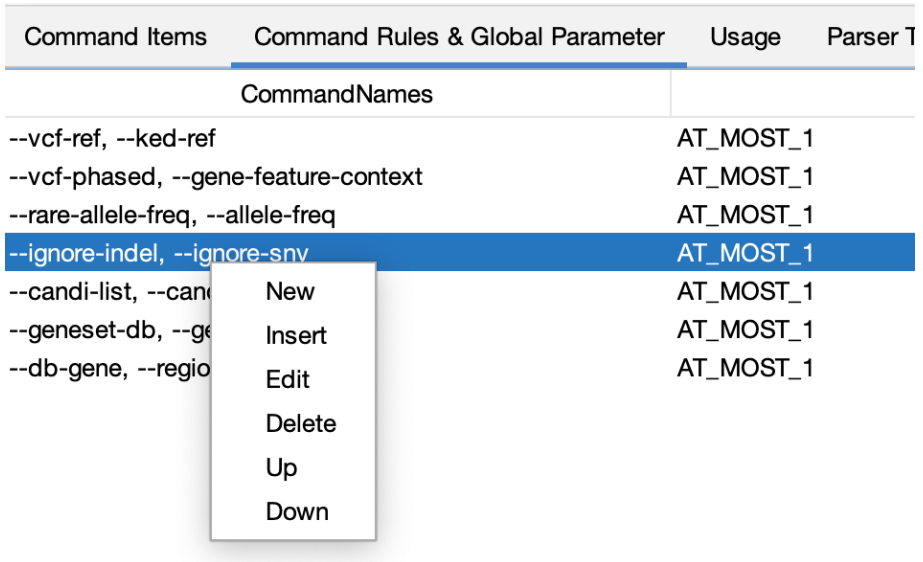
参数项面板下方搜索框处 (快捷键 Ctrl + F), 设置搜索的属性 (复选框) 及内容 (文本框), 按回车键查找符合的参数项并进行跳转、高亮显示. 搜索内容忽略大小写.



# 管理参数规则 (Command Rules)

在 Command Rules & Global Parameter 标签页中添加参数间的指令规则. 在参数规则面板空白处或选中参数规则点击鼠标右键, 展开管理菜单. 管理菜单包含以下 6 种操作:

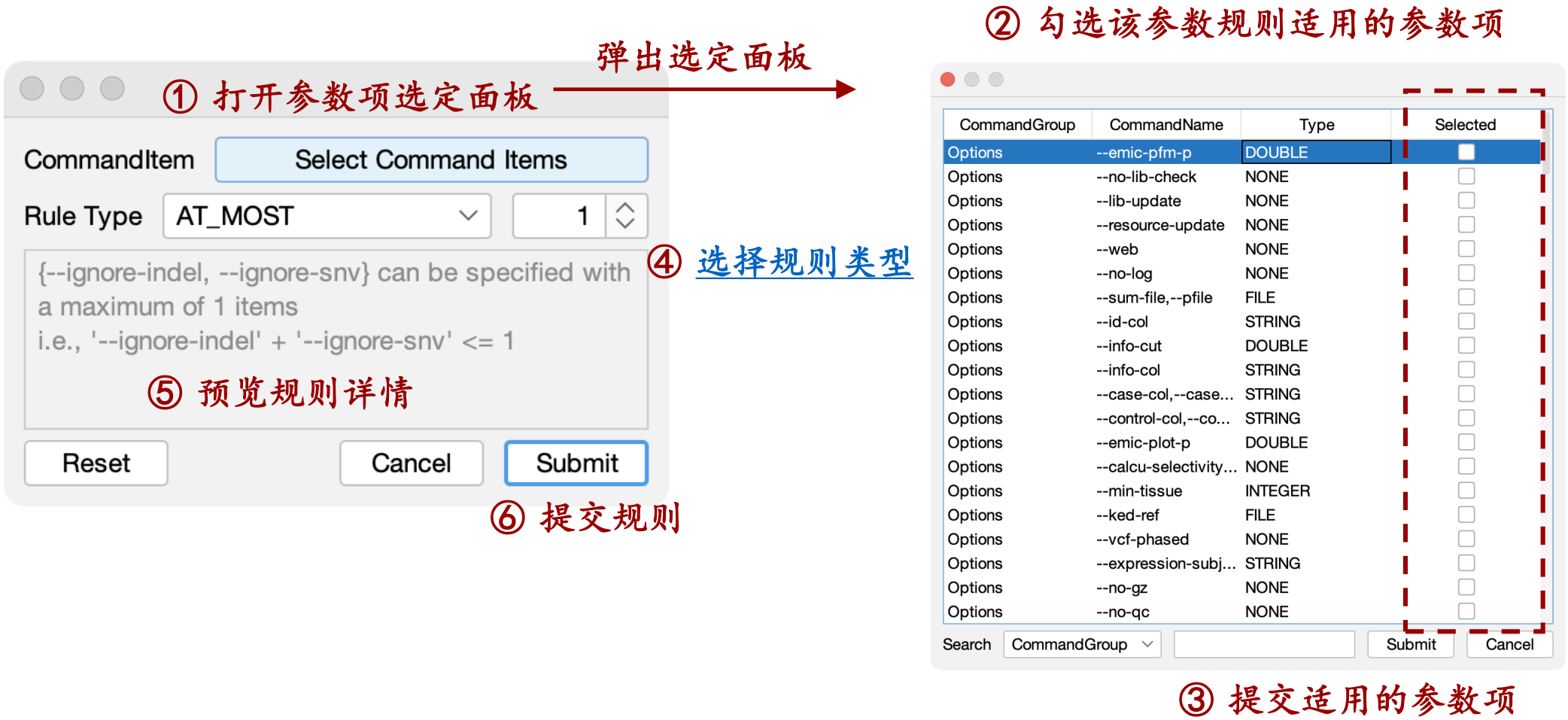
- **New:** 创建新参数规则 (快捷键: Ctrl + N)
- **Insert:** 在当前位置插入新参数规则
- **Edit:** 编辑该参数规则 (快捷键: 双击参数规则)
- **Delete:** 删除该参数规则 (快捷键: Ctrl + Delete)
- **Up:** 上移该参数规则 (快捷键: Ctrl + U)
- **Down:** 下移该参数规则 (快捷键: Ctrl + D)




下方菜单栏 "+" 对应 New 操作, "-" 对应 Delete 操作, "↑" 对应 Up 操作, "↓" 对应 Down 操作

# 编辑参数规则

新建或编辑参数规则时，弹出参数规则子面板. 无法为 “Help”, “Request” 类型参数设置规则.



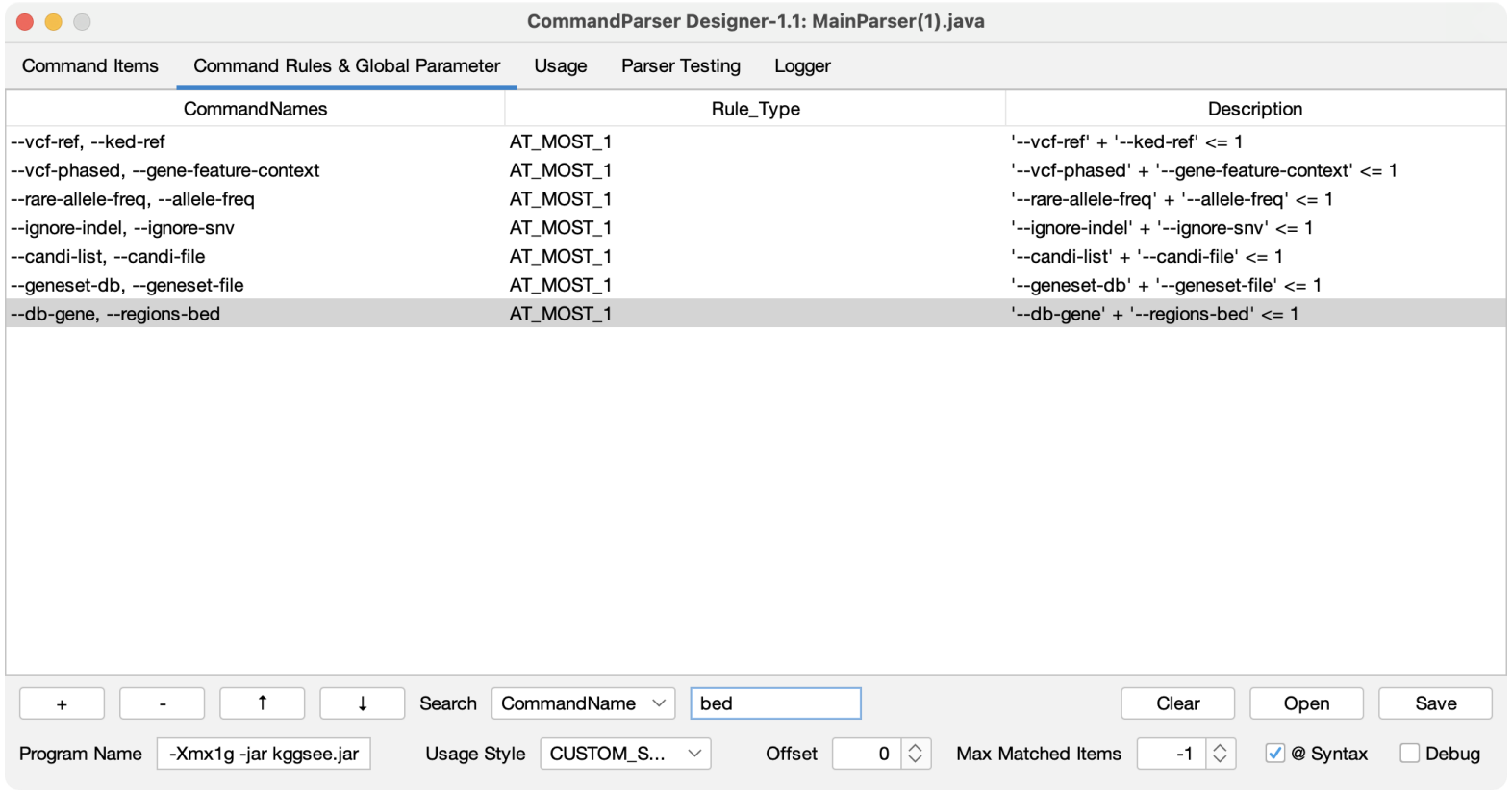
## 参数规则类型

当参数规则选定了参数项 $\{p_1, p_2, \cdots, p_n\}$ 时，参数规则对应的含义如下

| Command Rule | 条件数 | 描述 |
| --- | --- | --- |
| AT_MOST | $k$ | $p_1, p_2 \cdots, p_n$ 至多传入 $k$ 个 |
| AT_LEAST | $k$ | $p_1, p_2 \cdots, p_n$ 至少传入 $k$ 个 |
| EQUAL | $k$ | $p_1, p_2 \cdots, p_n$ 需要传入 $k$ 个 |
| MUTUAL_EXCLUSION | $k$ | $p_1, p_2 \cdots, p_k$ 与 $p_{k+1}, p_{k+2} \cdots, p_n$ 不能同时传入 |
| SYMBIOSIS | 不支持 | $p_1, p_2 \cdots, p_n$ 同时传入或同时不传入 |
| PRECONDITION | 不支持 | $p_1, p_2 \cdots, p_n$ 同时传入或同时不传入 |

# 搜索参数规则

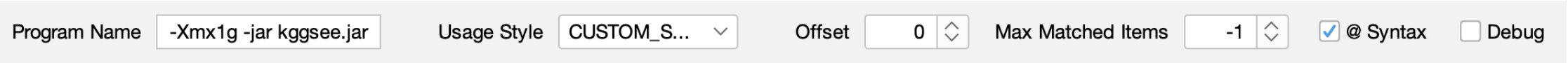
参数规则面板下方搜索框处 (快捷键 Ctrl + F), 设置搜索的属性 (复选框) 及内容 (文本框), 按回车键查找符合的参数规则并进行跳转、高亮显示. 搜索内容忽略大小写.



# 设置解析器全局参数

在 Command Rules & Global Parameter 标签页下部分设置解析器的全局参数. 全局参数包含:

- **Program Name:** 程序名.
- **Usage Style:** 自动化文档的格式. 双击复选框编辑格式, 选择 "..." 创建新文档格式.
- **Offset:** 参数偏移量.
  - 例如, 在 offset = 3 时, `bgzip compress <file> --level 5` 从 `--level` 处开始解析
- **Max Matched Items:** 最大匹配参数项个数, -1 表示不设置数量限制.
  - 传入的参数项达到最大个数时, 后续的参数不再解析, 而是作为最后一个匹配的参数项的值.
- **@Syntax:** @语法开关. 在@语法下, 识别 "@file" 类型参数, 该参数被替换为 file 文件内容.
- **Debug:** 调试模式开关. 标记为 "Debug" 的参数仅在调试模式下显示、可用; 在调试模式下将显示解析器工作日志.



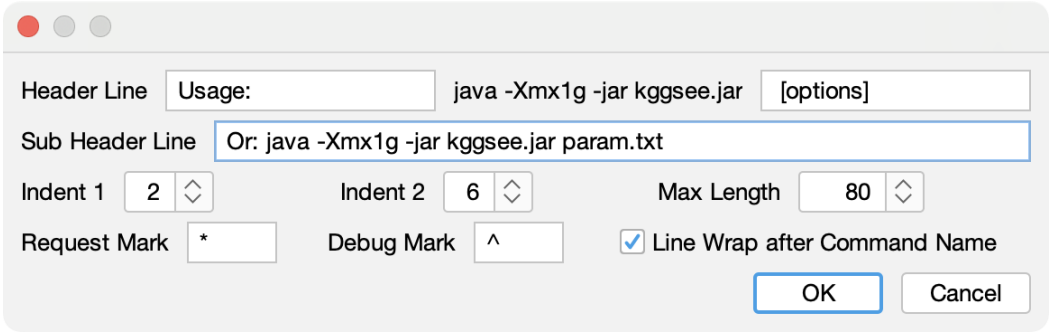
# 设置自动化文档的格式

Command Items | Command Rules & Global Parameter | Usage | Parser Testing | Logger

```
Usage: java -Xmx1g -jar kggsee.jar [options]
Or: java -Xmx1g -jar kggsee.jar param.txt
Options:
  --help,-help,-h
  --emic-pfm-p
      default: 2.5E-6
      format: --emic-pfm-p <double>
  --lib-update
  --resource-update
  --web
  --sum-file,--pfile
      path of the file storing GWAS summary statistics
      format: --sum-file <file> (Exists,File)
  --vcf-phased
  --geneset-db
      format: --geneset-db <string>
  --geneset-file
      format: --geneset-file <string>
  --geneset-enrichment-test
      default: 0.1
      format: --geneset-enrichment-test <double>
  --qqplot
      Draw quantile-quantile plot of p-values
Inputs/Outputs:
  --chrom-col
      column name of chromosome ID
      default: CHR
      format: --chrom-col <string>
```

Header Line 和 Sub Header Line

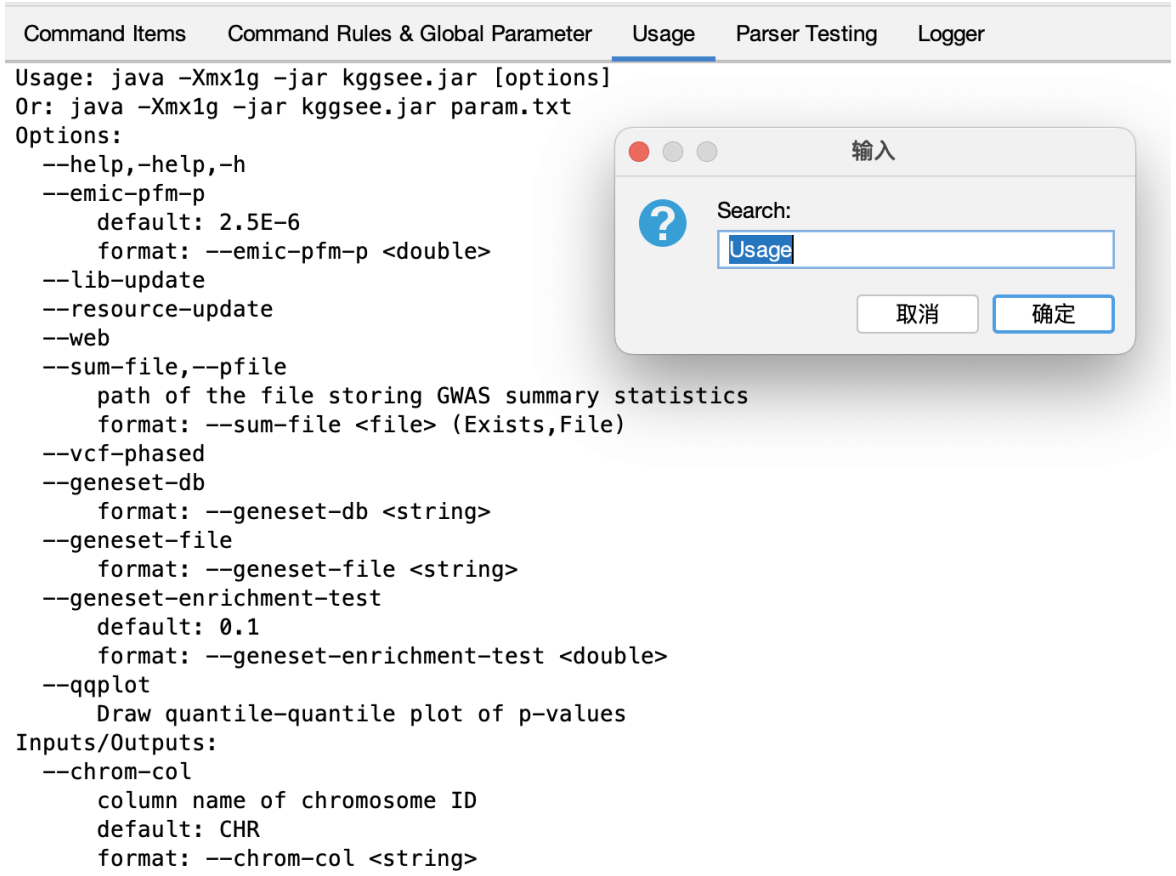
Max Length




Indent 1: 参数名前空格数
Indent 2: 描述文档前空格数
Max Length: 文档最大宽度 (超过宽度自动换行)
Request Mark: 必备参数前添加的标记
Debug Mark: 调试参数前添加的标记
Line Wrap after Command Name: 参数名后换行

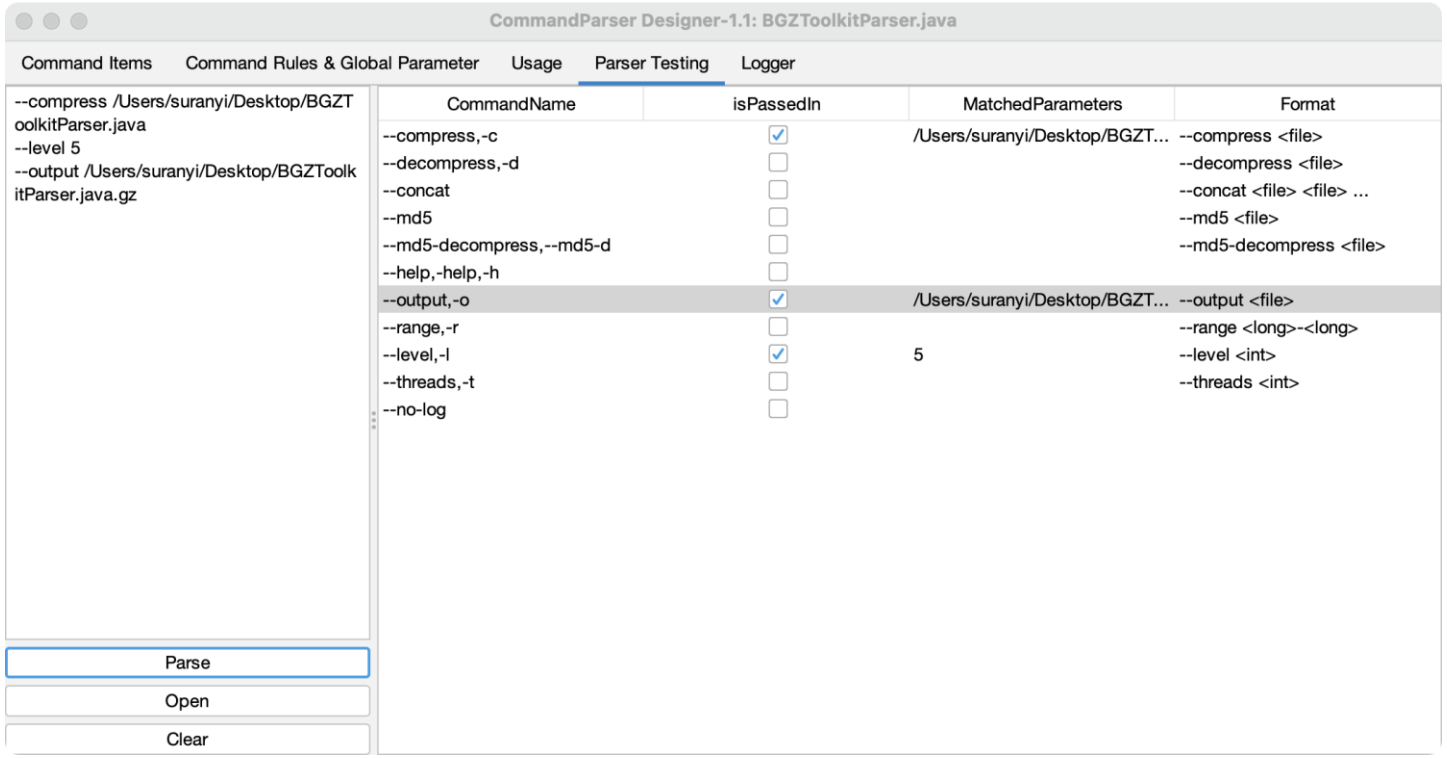
# 预览参数文档

在 Usage 标签页中预览解析器文档, 解析器文档可复制、可搜索 (快捷键: Ctrl + F).



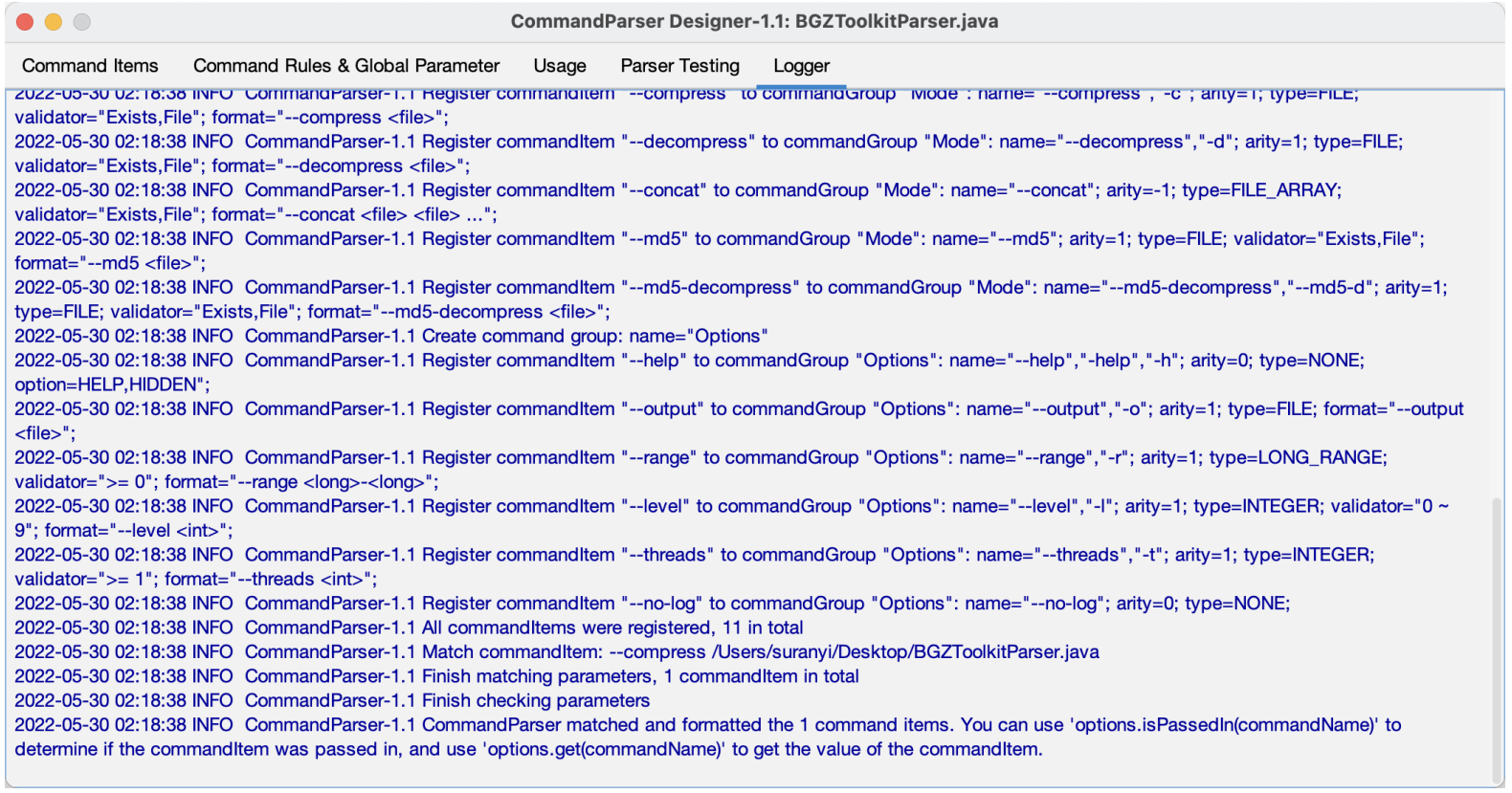
# 参数解析测试

在 Parser Testing 标签页中可进行参数解析测试. 在编辑框中输入参数, 并点击 “Parse” 按钮, 右侧将显示解析器对该次指令的参数捕获情况: isPassedIn (是否传入该参数), MatchedParameters (该参数项捕获的参数值), Format (输入格式). 双击参数项将跳转至 “Command Items” 标签页中的参数项位置.



# 查看日志

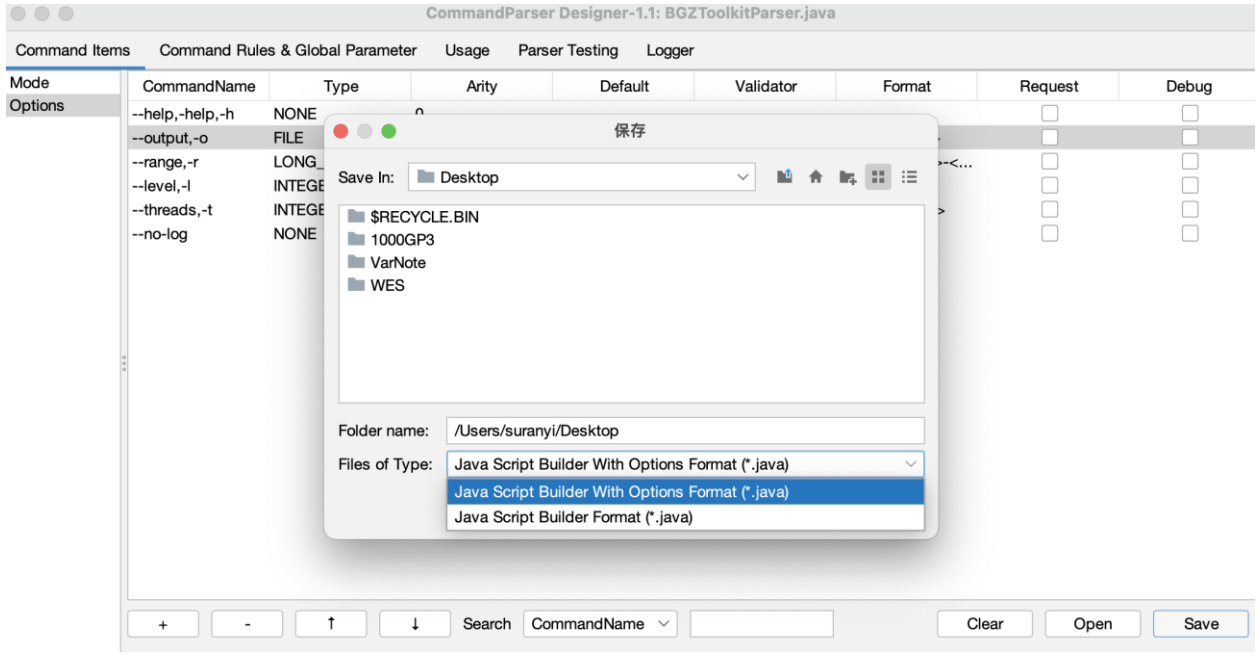
在 Logger 标签页中, 可以查看当前程序的工作日志. 该日志系统基于 log4j + Loggback 构建.



# 导出解析器

在 Command Items 标签页或Command Rules & Global Parameter 标签页中, 点击右下角 “Save” 按钮保存当前解析器 (快捷键: Ctrl + S). 提供两种保存格式:

- Java Script Builder With Options Format: 解析器及参数项构建文件
- Java Script Builder Format: 解析器文件



# Java Script Builder With Options Format

Java Script Builder With Options Format 根据参数项的主参数名 (Command Name) 推断变量名 (驼峰命名法), 根据参数项的参数类型 (Command Type) 设置参数类型, 无需对变量进行类型转换.

```java
public static void main(String[] args) throws IOException {
    if (args.length == 0) {
        // 没有传入参数时, 打印文档
        System.out.println(Parser.usage());
        return;
    }

    Parser options = Parser.parse(args);
    if (options.help.isPassedIn) {
        // 传入 help 指令时, 打印文档
        System.out.println(Parser.usage());
        return;
    }

    // 业务逻辑
    HttpDownloader.instance(args[0])
            .setOutputFile(options.output.value)
            .setThreads(options.threads.value)
            .setPrintLog(!options.noLog.isPassedIn)
            .setTempDir(options.tempDir.value)
            .setProxy(options.proxy.value)
            .setTimeOut(options.timeout.value)
            .clean(options.overwrite.isPassedIn)
            .download();
}
```

在入口函数使用解析器
桥接参数指令与业务逻辑

| API 方法 | 描述 |
| --- | --- |
| options.变量名.value | 获取参数值（未传入时，该值为默认值） |
| options.变量名. isPassedIn | 该参数项是否被传入 |
| options.变量名. matchedParameter | 捕获的参数值（字符串原始格式） |
| Parser.usage() | 获取解析器文档 |
| Parser.getParser() | 获取解析器对象 |

# Java Script Builder Format

Java Script Builder Format 创建解析器单例, 通过变量名 (参数项的任一变量名) 访问参数解析信息 (值、是否被传入、捕获值), 在获取值时需要进行格式转换.

```java
public static void main(String[] args) throws IOException {
    if (args.length == 0) {
        // 没有传入参数时, 打印文档
        System.out.println(Parser.usage());
        return;
    }

    CommandOptions options = Parser.parse(args);
    if (options.isHelp()) {
        // 传入 help 指令时, 打印文档
        System.out.println(Parser.getParser());
        return;
    }

    // 业务逻辑
    HttpDownloader.instance(args[0])
            .setOutputFile((File) options.get("--output"))
            .setThreads((int) options.get("--threads"))
            .setPrintLog(!options.isPassedIn("--no-log"))
            .setTempDir((File) options.get("--temp-dir"))
            .setProxy((String) options.get("--proxy"))
            .setTimeOut((int) options.get("--time-out"))
            .clean(options.isPassedIn("--overwrite"))
            .download();
}
```

在入口函数使用解析器
桥接参数指令与业务逻辑

| API 方法 | 描述 |
| --- | --- |
| options.get(参数名) | 获取参数值（未传入时，该值为默认值）<br>获得的参数类型是 Object类型，需要手动转换格式 |
| options.isPassedIn(参数名) | 该参数项是否被传入 |
| options.<br>getMatchedParameter(参数名) | 捕获的参数值（字符串原始格式） |
| Parser.usage() | 获取解析器文档 |
| Parser.getParser() | 获取解析器对象 |

# 重编辑解析器文件

导出的解析器文件是 Java 源代码文件 (如右图), 用户可以直接修改 Java 源代码文件实现解析器的修改. 此外,设计器图形界面也支持导入该源码文件 (拖拽文件到界面窗口、点击 Open 按钮或快捷键 Ctrl + O) 实现重编辑.

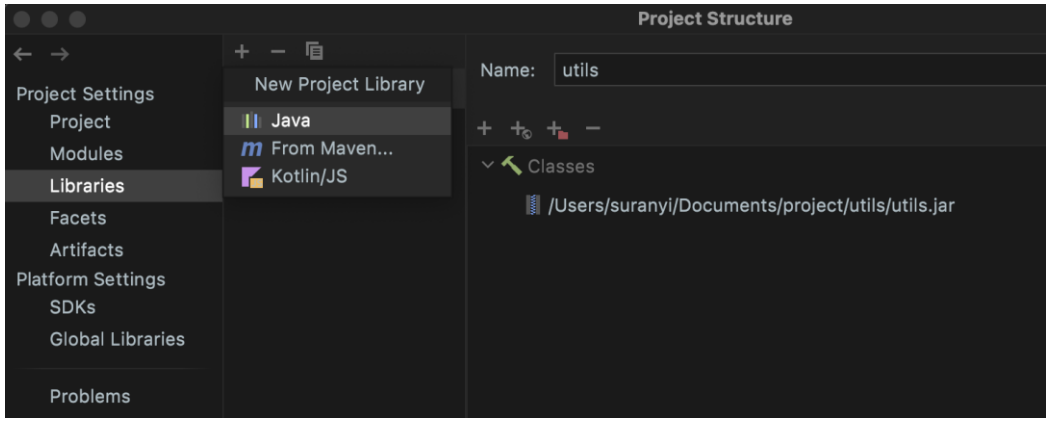
CommandParserDesigner 将解析器源代码文件动态编译为 class 文件, 并通过 .getParser() 获取解析器对象. 最后, 设计器图形界面根据解析器对象的成员信息复现解析器.

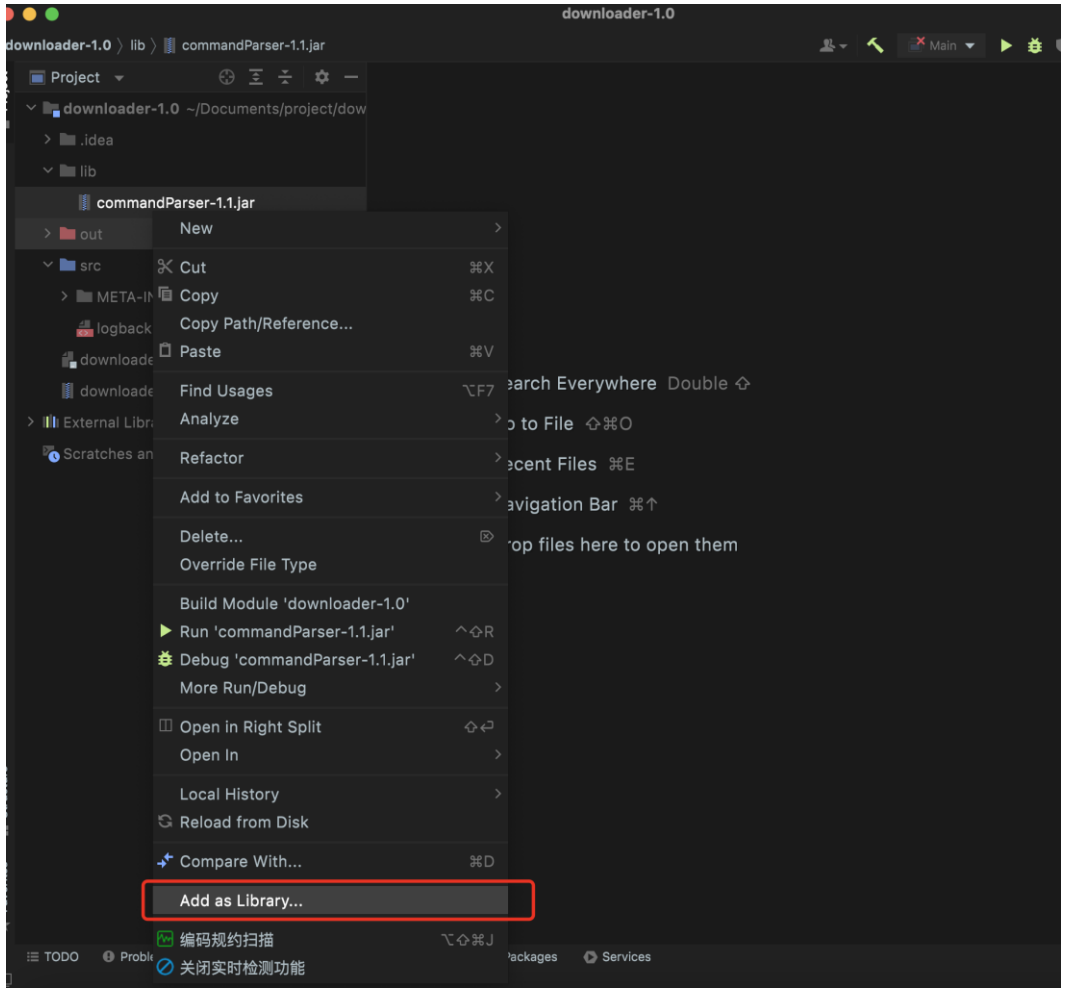
```
parser.setProgramName("<mode>");
parser.offset(0);
parser.debug(false);
parser.usingAt(true);
parser.setMaxMatchedNum(-1);
parser.setUsageStyle(DefaultStyleUsage.UNIX_TYPE_1);

CommandGroup group001 = parser.addCommandGroup( groupName: "Mode");
group001.register(FILE.VALUE,  ...commandNames: "--compress", "-c")
        .validateWith(FILE.validateWith( checkIsExists: true,  checkIsFile: true))
        .setDescription("Compression using parallel-bgzip (supported by CLM algorithm).");
group001.register(FILE.VALUE,  ...commandNames: "--decompress", "-d")
        .validateWith(FILE.validateWith( checkIsExists: true,  checkIsFile: true))
        .setDescription("Decompress or recompress partial (or full) bgzip file.");
group001.register(FILE.ARRAY,  ...commandNames: "--concat")
        .arity(-1)
        .validateWith(FILE.validateWith( checkIsExists: true,  checkIsFile: true))
        .setDescription("Concatenate multiple files.");
group001.register(FILE.VALUE,  ...commandNames: "--md5")
        .validateWith(FILE.validateWith( checkIsExists: true,  checkIsFile: true))
        .setDescription("Calculate a message-digest fingerprint (checksum) for input file.");
group001.register(FILE.VALUE,  ...commandNames: "--md5-decompress", "--md5-d")
        .validateWith(FILE.validateWith( checkIsExists: true,  checkIsFile: true))
        .setDescription("Calculate a message-digest fingerprint (checksum) for decompressed file.");
```

# 导入 CommandParser-1.1.jar

**方式一 (下图):** 在项目结构管理中选择 “Libraries”，添加 commandParser-1.1.jar.

**方式二 (右图):** 在工程项目中创建 lib 文件夹，添加 commandParser-1.1.jar，并右键点击 “Add as Library...”.



# 创建 Jar 包

创建解析器源代码文件、导入 commandParser-1.1.jar 包、编写入口函数后，就完成了完整的命令行程序开发. 最后将 Java 工程项目打包为 jar 包 (以 IDEA 为例):

依次点击: Project Structure... > Artifacts > + > JAR > From modules with dependencies ... 显示左图窗口. 在 Main Class 处选择入口函数位置，点击 OK. 最后，点击 Build > Build Artifacts 构建 jar 包.