第二章 CPLEX 基础

2.1 CPLEX 概述

2.1.1 CPLEX 简介

CPLEX 是 IBM 公司开发的一款商业版的数学优化求解器，也被称为 CPLEX Optimization Studio。CPLEX 专门用于求解各种数学优化问题，如线性规划(LP)、混合整数规划(MIP)、二次规划(QP)等。CPLEX 以其高效求解能力、灵活建模功能和广泛的应用领域，成为优化领域的重要工具之一。

由于 CPLEX 最初是由 Robert Bixby 博士在 1987 年共同创立的 CPLEX Optimization, Inc.开发的，1997 年被 ILOG 公司收购，2009 年 IBM 公司收购了 ILOG 公司，因此 CPLEX 又称为 IBM ILOG CPLEX。为了描述简便性，后续通常直接简称其为 CPLEX。

CPLEX 作为一款商业化的数学优化软件包，具有许多显著的优点，这些优点使其在优化领域占据了重要的地位。CPLEX 主要优点有：

（1）高效的求解算法：CPLEX 采用了多种优化算法和启发式方法，能够快速求解大规模的优化问题，其高效的求解能力使得用户能够迅速得到问题的解决方案。

（2）灵活的建模功能：CPLEX 支持多种优化建模语言和接口，包括 C、C++、Java、Python 等，这使得用户可以在其他应用系统集成 CPLEX 求解功能。

（3）强大的功能：CPLEX 支持各种复杂约束和目标函数，能够处理大规模优化问题，其强大的功能使得用户能够解决复杂的优化问题，并得到高质量的解决方案。

（4）健壮而可靠：CPLEX 经过了大量的安装使用和测试，其优化求解性能得到不断的提升和验证。

（5）广泛的应用领域：CPLEX 被广泛应用于供应链管理、生产调度、资源分配、网络设计、金融投资等领域，它帮助用户做出最优决策和规划，提高资源利用效率、降低成本并提升盈利能力。

（6）可视化和交互性：CPLEX 提供了直观的图形界面和交互式工具，帮助用户分析问题、调整参数和优化求解过程，这使得用户能够更加方便地使用 CPLEX，提高求解效率。

综上所述，CPLEX 以其高效的求解算法、灵活的建模功能、强大的功能、健壮而可靠的性能、广泛的应用领域、良好的可视化和交互性以及可扩展性和灵活性等优点，在优化领域占据了重要的地位。这些优点使得 CPLEX 成为解决各种数学优化问题的首选工具之一。

2.1.2 CPLEX 版本

CPLEX 软件有试用版、学术版和商业版三种版本的安装包，主体功能都提供数学规划问题的求解功能并支持多种编程语言的集成调用，但是三个版本还具有如下的区别：

（1） 试用版为免费版，通常限制 1000 个变量/约束，主要用于用户评估 CPLEX 是否满足其应用需求以及进行小规模问题的测试；

（2） 学术版为免费版，变量和约束数量无限制，但是需要通过学校官方邮件进行认证后方可免费使用，该版本广泛应用于学术界的研究和教学活动；

（3） 商业版需要购买许可证，变量和约束数量无限制，且 IBM 公司提供全面的技术支持和服务，适用于商业环境，满足企业对大规模和复杂优化问题的求解需求。

用户根据需要从 IBM 公司网站上选择试用版和学术版软件下载，并进行安装即可。

2.2 CPLEX 项目及示例

2.2.1 CPLEX 项目介绍

使用 CPLEX 的方法有两种：直接在 CPLEX Optimization Studio 集成开发环境（IDE）中进行项目开发，在 C、C++、Java 等语言中进行集成开发。本书介绍直接在 CPLEX IDE 中进行项目开发和运行等操作过程，在该开发环境中开发优化模型使用的 CPLEX 专用的优化编程语言（Optimization Programming Language， OPL），因此这类项目也称为 OPL 项目。单个 OPL 项目主要由模型文件（.mod）、数据文件（.dat）和设置文件（.ops）构成。单个 OPL 项目可以只包含单个模型文件，而不包含数据文件和设置文件，也可以包含多个模型文件、数据文件和设置文件，不过这些文件之间的关系需要由运行配置来维护。CPLEX 项目各类文件和 CPLEX IDE 界面基本如图 2.1 所示，其中左侧为项目所包含的给类文件，左下角为求解后获得的各个变量值，中间部分为对 mod、dat 和 ops 进行编辑的界面，右侧显示当前编辑文件的大纲内容，右下方为优化求解的过程和结果显示内容。

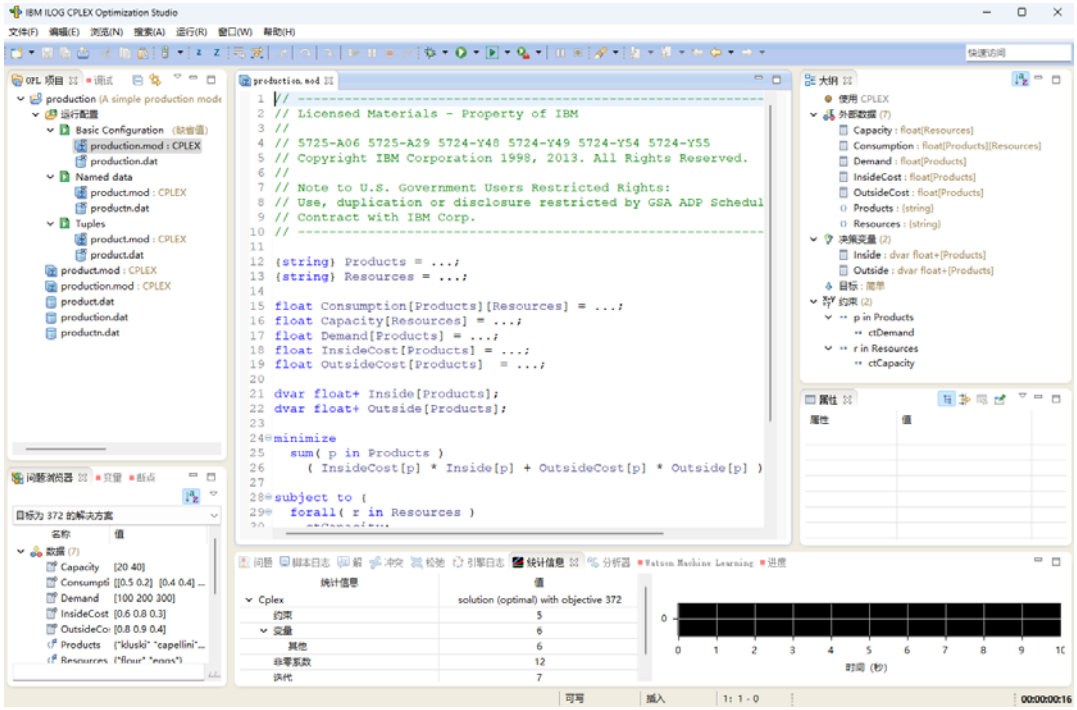


图 2.1 CPLEX IDE 界面截图

下面对 OPL 项目三类文件进行简要说明。

（1）OPL 项目模型文件

OPL 项目模型文件是以 mod 为扩展名的文件，该类文件包含建模和脚本语句，主要包

括数据的声明、决策变量的声明、目标函数和约束等内容。

- 数据的声明：数据的声明是对模型中涉及的变量进行命名，这样就可以在模型中调用数据文件中的数据。注意：若在数据声明的同时列出数据，此时该变量不再需要数据文件；若在数据声明时使用了省略符“…”来表示数据，那么该变量对应的数据必须在数据文件中列出来。
- 决策变量的声明：决策变量的声明是为模型中每个变量命名并明确其类型。
- 目标函数：目标函数就是该项目中需要进行优化的函数，该函数必须由上述模型文件中声明的变量和数据组成。
- 约束：约束是指模型可行解所需要满足的条件，在 subject to 中进行声明。

（2）OPL 项目数据文件

OPL 项目数据文件是以 dat 为扩展名的文件，该类文件储存模型中的数据。若模型复杂，则一个 OPL 项目可以由一个或多个数据文件来储存模型中的数据。将模型文件和数据文件分开存储，可以更好地对大型问题进行求解，条理更加清晰。

（3）OPL 项目设置文件

OPL 项目设置文件是以 ops 为扩展名的文件，该类文件包含 CPLEX 参数、CP Optimizer 参数和 OPL 语言选项。CPLEX 参数选项包括对内点算法、Benders 算法、分布式混合整数规划、网络、并行优化、预处理（聚集器和预求解器）、筛选、单纯形、解法完善、解法池、容错、调整参数、限制、显示和输出、诊断等方面的设置，可以根据模型需要自行设置。多数情况下保持系统缺省设置即可。

（4）OPL 项目运行配置

OPL 项目运行配置是设置和处理项目运行过程中模型文件、数据文件和设置文件相互关系的方法。通常用于同一类问题使用多种类型的模型并求解多种规模算例数据时的分类管理，例如某个问题可以使用三个数学模型来进行表示和求解，为了验证这三个模型的运算性能，将分别测试这三个数学模型对 5 个规模算例的优化求解，这是则可以在同一个 OPL 中设置多个运行配置分别进行算例实验。

2.2.2 CPLEX OPL 项目示例

（1）示例描述

某公司生产车间有两台加工设备 A 和 B 以及一个测试台 C 可进行四种家电产品 I、II、III 和 IV 的生产。已知生产这两种产品单件所需设备 A、设备 B 和测试台 C 的时间，下个月 A、B 设备和调试台 C 可用于生产这四种家电的工时、售出单件各种产品的获利如表 2-1 所示。问该公司下个月应生产这四种家电各多少件，可实现获取的利润最大。

该决策问题为典型的线性规划问题，且为整数线性规划问题，可以使用 CPLEX 进行求解。在使用 CPLEX 进行建模和求解之前，需要设计问题的数学模型。这里设计示例问题的数学模型如下：

目标函数：

$$\max \qquad z = 3x_1 + 2x_2 + 2x_3 + 4x_4 \qquad (2\text{-}1)$$

约束条件：

$$5x_2 + 2x_3 + x_4 \le 15 \qquad (2\text{-}2)$$

$$6x_1 + 2x_2 + 3x_3 + 5x_4 \le 220 \qquad (2\text{-}3)$$

$$x_1 + x_2 + x_3 + x_4 \le 80 \qquad (2\text{-}4)$$

$$x_i \in I^+ \quad \forall i = 1,2,3,4 \qquad (2\text{-}5)$$

决策变量 x_1, x_2, x_3, x_4 分别表示该公司生产四类产品数量，目标函数（2-1）表示生产四种产品所获得的最大利润，约束条件（2-2）、（2-3）和（2-4）分别表示四种产品生产数量受设备 A、B 和测试台 C 可用工时的限制，约束（2-5）表示四类产品生产数量的非负整数约束。

表 2-1 示例问题相关数据表

项目	I	II	III	IV	月可用工时
设备 A(h)	0	5	2	1	100
设备 B(h)	6	2	3	5	220
测试台 C(h)	1	1	1	1	80
利润(元)	3	2	2	4	

（2）新建项目

单击“文件”菜单，选择“新建”菜单项，选择“OPL 项目”打开新建项目窗口，菜单路径如图，如图 2-2 所示。

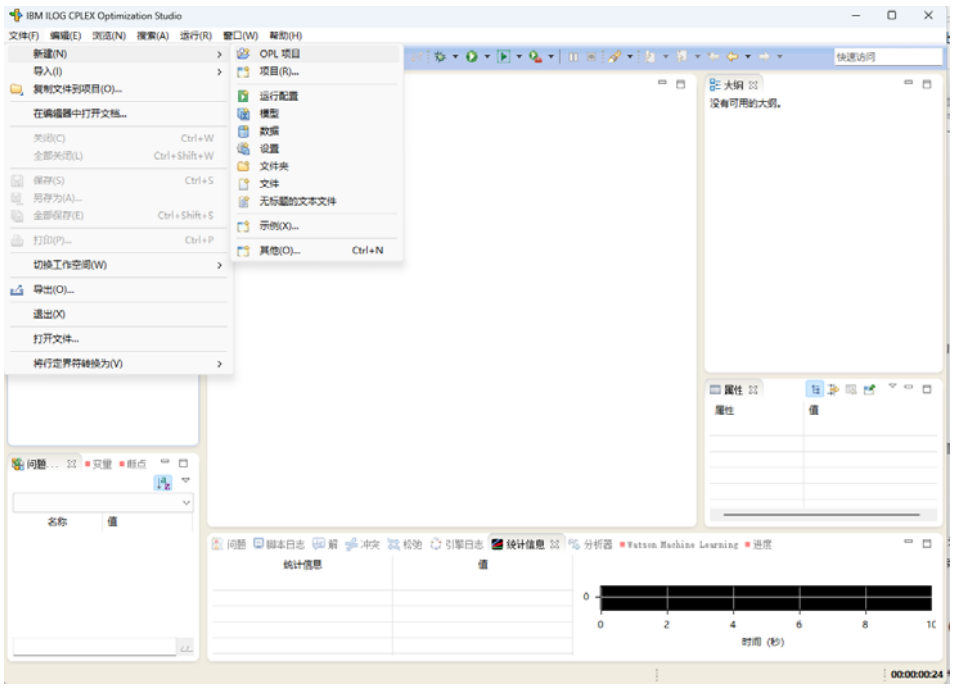


图 2-2 创建新的 OPL 项目菜单操作

新建项目窗口如图 2-3 所示，在“项目名称：”后的文本框输入 OPL 项目名称，并设置项目文件存放位置，然后将选项中的四个内容都选中，在“描述”后文本框输入项目的简要说明信息，最后单击“完成”则完成项目的初步创建。

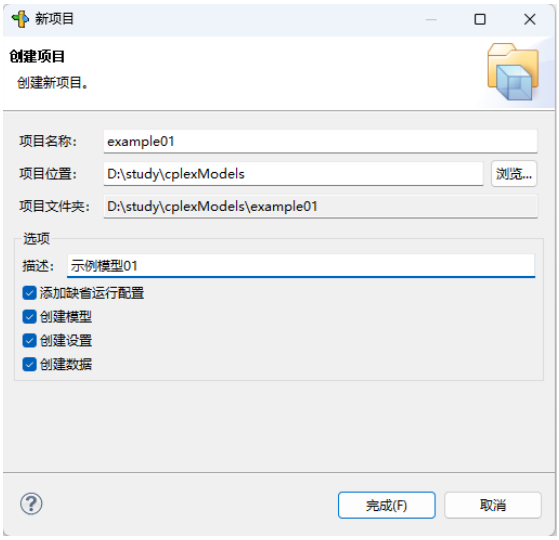


图 2-3 创建 OPL 项目命名和选项设置界面

注意：项目名称命名不要包含中文字符、空格等特殊字符，首字母必须为 26 个英文字母中的一个，以防止运行出错。

此时在 CPLEX 软件左侧项目列表中便有了一个空的 OPL 项目，包含运行配置、运行文件（example01.ops）、模型文件（example01.mod）、数据文件（example01.dat）和一个运行配置（配置 1），如图 2-4 所示。“配置 1”中包含刚才创建的模型、设置文件和数据文件。

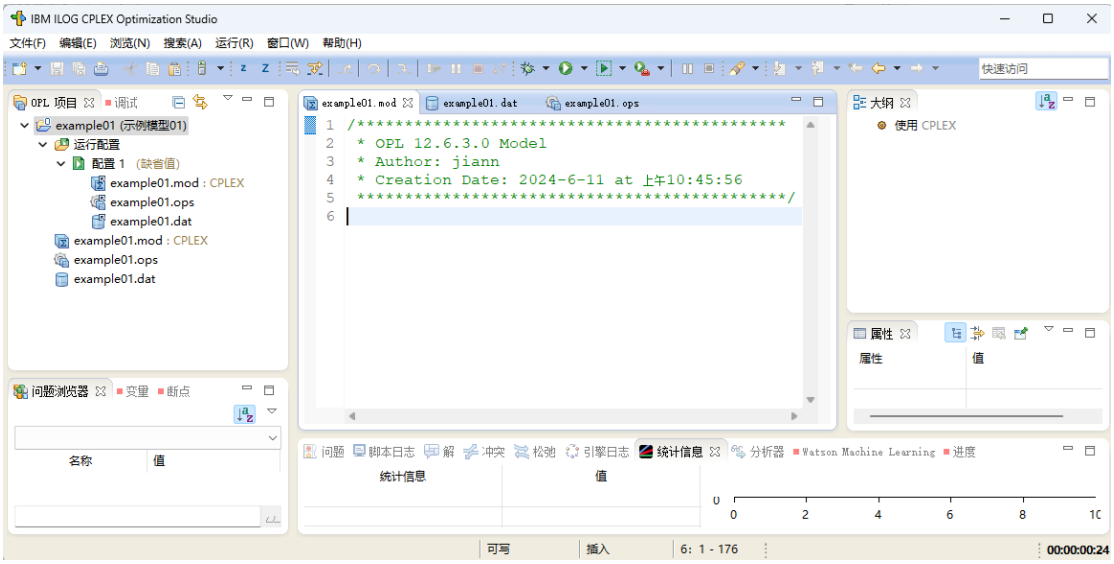


图 2-4 创建完成后的空 OPL 项目结构

在窗口中间的 example01.mod 编辑窗口根据示例的数学模型输入 OPL 语言模型，单击“保存”按钮即可保存该文件，如图 2-5 所示。

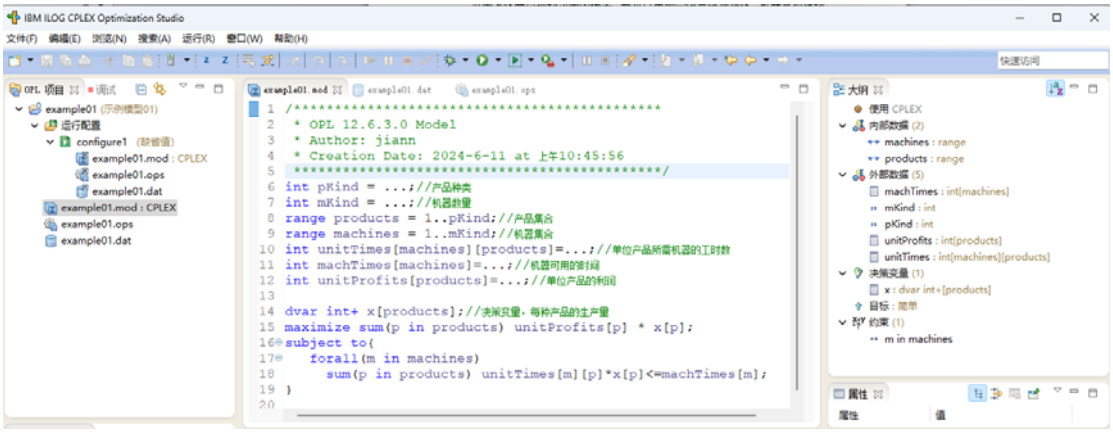


图 2-5 示例模型文件设计界面

示例模型文件（example01.mod）中第 6-12 行代码为问题参数设计，int 为整数型参数的定义，等号右侧三个点号（…）表示该参数的值由数据文件指定；第 14 行是决策变量 x 的定义；第 16 行是目标函数的定义，该行中 maximize 表示优化运算是为了获得后续表达式的最大值，sum 为求和符号，是 OPL 对式（2-1）的精炼表达方式；行 16-19 为约束条件，使用 subject to 标识，问题全部约束都放在 subject to 的大括号以内，示例模型中约束条件（2-2）、（2-3）和（2-4）分别为全部产品所用三台机器工时的约束，可以直接使用行 17-18 这种方式表示即可；式（2-5）决策变量非负整数约束已经在行 14 中进行了限定了，不需要在约束条件中再进行约束。

双击 example01 项目下的 example01.dat 文件激活中间部分的编辑窗，将示例数据输入到该数据文件，模型中以…方式赋值的变量和数组都需要在该数据文件中进行指定，具体如图 2-6 所示。

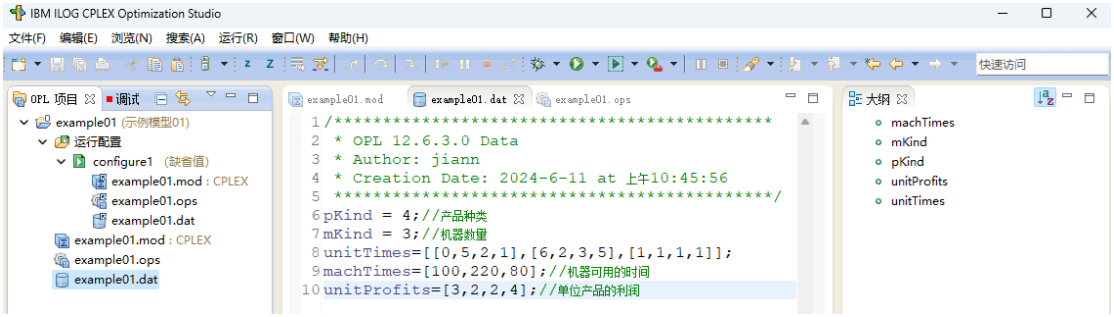


图 2-6 示例数据文件设计界面

（3）执行运行配置

若在建立模型的时候没有更改运行配置名称，那么执行运行配置前要先将运行配置名称更改为没有中文的名称，否则执行时会出现错误。在案例模型中，将运行配置名称更改为 configure1，由于原有配置文件下包含了模型和数据文件，直接按照图 2-7 所示，在运行配置 configure1 上点鼠标右键选择“运行这个”将执行该模型优化运算。

若在创建文件时（参考图 2-3）未勾选添加缺省运行配置、创建模型、创建设置、创建数据选项，此时将期望运行的模型文件（example01.mod）和数据文件（example01.dat）拖

入到运行配置中即可。

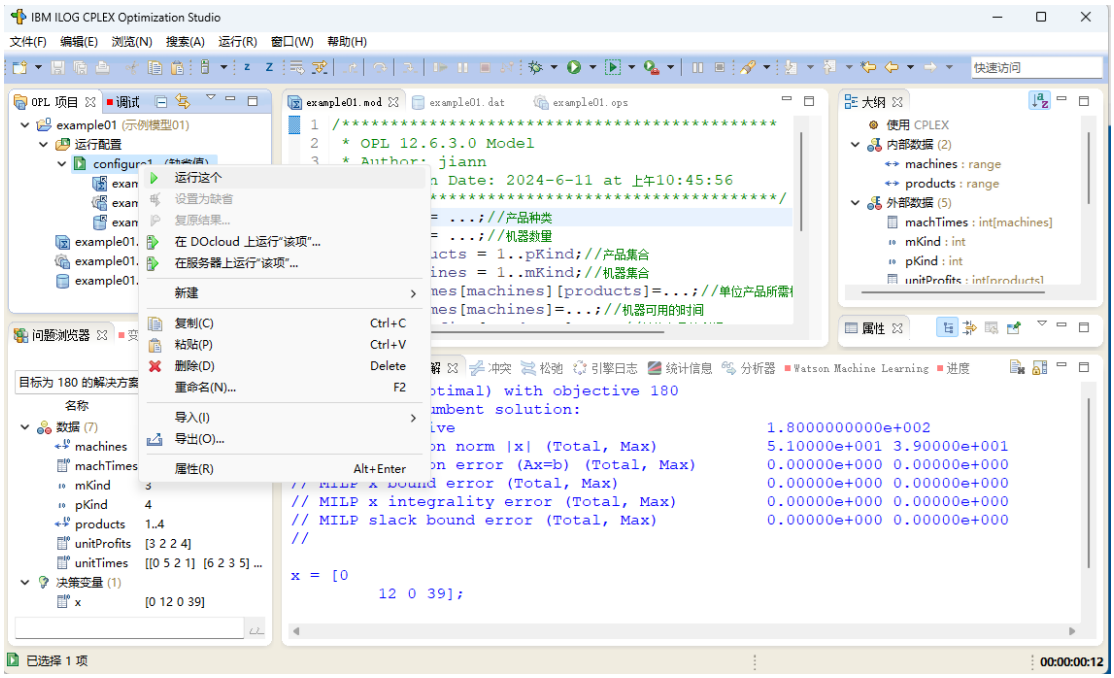


图 2-7 执行运行配置界面

图 2-7 右下部分给出 OPL 项目运行配置执行完毕后的结果，结果显示若干输出选项卡，分别为问题、脚本日志、解、冲突、松弛、引擎日志、统计信息、分析器等选项卡。图 2-7 显示“解”选项卡的内容，可以看出最优目标值为 180，决策变量 x 取值依次为 0、12、0、39，即生产 12 个单位的产品 II 和 39 个单位的产品 IV 可实现最大利润 180。

（4）保存和恢复结果

在运行结束之后，若关闭 CPLEX Studio 软件，则下次再打开 CPLEX Studio 时需要重新运行配置才可以查看结果。因此及时将求解结果进行保存，方便下次打开 CPLEX Studio 软件时查看之前运行过的结果。

保存求解结果、历史记录的步骤为：在菜单栏中选择“运行”，单击“保存结果”菜单项即可。该操作将结果保存为问题浏览器中所示内容（数据、决策变量值、表达式和约束）、模型文件和日志文件（脚本日志、角犀、冲突、松弛、引擎日志、统计信息和分析器）。

结果保存后，单击“example01”项目的“运行配置”，可以看出“configure1”图标右下角会有一个时钟的标志，表示结果已经保存，如图 2-8 所示。

图 2-8 结果保存后的状态

如果要恢复保存的结果可以在对应运行配置名称上点右击，选择“复原结果”即可。

2.3 CPLEX OPL 常用符号

CPLEX 模型使用 OPL 来进行描述，其中常用的符号主要是运算符，运算符又分为算术运算符、关系运算符、逻辑运算符、集合运算符和其他符号。

2.3.1 算术运算符

算术运算符包括+、-、*、/、^、%(或者 mod)、div 这 7 种符号，具体含义和用法如表 2-2 所示。

表 2-2 算术运算符

运算符名称	OPL 中运算符表示	举例
加法运算符或者正值运算符	+	1+2 dvar float+ x[products]
减法运算符或者负值运算符	-	3-2 -9
乘法运算符	*	2*3
除法运算符	/	4/2
幂运算符	^	2^3
取余运算（或模运算）	%或者 mod	4%3（4 mod 3）
整除运算	div	5 div 2

2.3.2 关系运算符

关系运算符用来表示关系表达式，如优化问题中的约束条件，包括==、!=、>、>=、<、<=，关系运算符如表 2-3 所示。

表 2-3 关系运算符

运算符名称	OPL 中运算符表示
相等	==
不等	!=
大于	>
大于等于	>=

小于	<
小于等于	<=

注意：

（1）区分关系运算符和关系表达式，*x*>0 是一个关系表达式,“>”是一个关系运算符。

（2）注意区别“==”和“=”,“==”是一个关系运算符，“=”是一个赋值运算符。

（3）一个关系表达式只能含有一个关系运算符，如关系表达式 x<=y<=z 在 OPL 中无效，需要表示成 x<=y，y<=z 这两个表达式。

2.3.3 逻辑运算符

逻辑运算符是根据表达式的值来返回真值或假值。在 OPL 中，认定非 0 为真值，0 为假值。逻辑运算符如表 2-4 所示。

表 2-4 逻辑运算符

运算符名称	OPL 中运算符表示
与	&&
或	||
非	!

（1）与

“与”运算符使用“&&”来表示，是双目运算符。两个条件如果同时成立，则结果为“真”，否则结果为假。当该运算符前后两个条件的逻辑值都为“真”时，整个达式的结果为“真”，值为 1；否则为“假”，值为 0。

例如：int x=3,y=-10,z=0；其中 x 非零，为真；y 非零，为真；z 值为零，为假。因此 x&&y 为真，值为 1，x&&z 或 y&&z 为假，值为 0。

（2）或

“或”运算符使用“||”来表示，是双目运算符。两个条件中只要有一个成立，则结果为“真”。当运算符前后两个条件的逻辑值有一个为“真”时，整个表达式的结果为“真”，值为 1；两个条件的逻辑值都为“假”时结果为“假”，值为 0。

例如：int x=3,y=-10,z=0; 则 x||y 或 x||z 或 z||y 都为真，值为 1；z||0 的值是 0。

（3）非

“非”运算符使用“!”来表示，是单目运算符，取反的意思。使用“!”运算符后，如果条件是真，则结果为假；如果条件是假，结果为真。

例如：int x=-3，y=!x，z=!y，那么最后 y=0，z=1。

2.3.4 集合运算符

OPL 中有 4 个集合运算符，分别为：sum、prod、min 和 max。

（1）sum

sum 表示求和运算，其语法为：

```
sum (X in Y)  表达式;
或   sum (X in Y :条件表达式)  表达式;
```

表示对 Y 中的所有元素按照表达式进行求和。例如：

```
minimize
   sum (o in SCities, d in DCities)
      Cost[o][d] * trans[o][d]; //对 scities、DCities 中的所有元素求总运费之和

forall(o in SCities)
   ctSupply:
   sum(d in DCities)
   trans[o][d]==Supply[o];//对 DCities 中的所有元素求城市 o 的总运出量之和，形成城市 o 供应量的平衡约束。
```

（2）prod

prod 表示乘积运算，其语法为：

```
prod（限定符）表达式;
```

表示满足限定符（限定符是指用于限定类型和类型成员的声明）内的集合的所有表达式相乘，且规定 prod（空集）=1。

例如：

```
int n=6;
int factorial = prod(i in 1..n) i;
assert(factorial == 720);
```

这个例子中:

int n=6，定义了一个整数变量。

int factorial = prod (i in1..n) i，定义整数变量 factorial，且其值为 1-6 这 6 个整数的乘积（即 720）。

assert (factorial == 720)，是一个断言函数，判断其乘积是否等于 720，所以这个断言函数是正确的。

（3）min

min 表示最小值运算，其语法为：

```
min (X in Y)  表达式;
```

表示 Y 中所有元素按照“表达式”计算后的最小值。

例如：

```
{string}places = ...;
float prices[places] = ...;
float minprice = min(i in prices) i;
```

（4）max

max 表示最大值运算，其语法为：

```
max (X in Y)  表达式;
```

表示 YY 中所有元素按照“表达式”计算后的最大值。

例如：

```
float maxprice = max(i in prices) i;
```

又如：

```
dvar int x[1..20] in 0..20; //x 为取值在 0 至 20 之间数值的数组型决策变量
minimize max(i in 1..20) x[i];
```

2.3.5 其他符号

（1）赋值符号“=”

在模型文件及数据文件的数据定义中使用，各种类型数据的定义都需要用到赋值符号。

（2）空格

在模型文件中空格用来分隔关键字，但是不能分隔数据；在数据文件中空格用来分隔数据。

使用空格可使 CPLEX 代码的可读性更高。

（3）方括号“[]”

方括号可以作为下标运算符和初始化数值型数据数组。

（4）括号“()”

括号表示函数命令范围限制。

（5）尖括号“<>”

尖括号在初始化数据时用来存储每个元组中的元素。

（6）大括号“{}”

大括号的作用：

① 表示字符串赋值序列的起始范围；

② 用于初始化字符型数据{string)；

③ 用于约束条件 execute {}；

④ 用于流控制模块：{int}。

（7）分号“;”

分号可用于目标函数、每一个约束条件、每一个数据定义、每一个决策变量定义的结尾，表示此行代码结束。

（8）逗号“,”

在模型文件和数据文件中逗号用于分隔数据，主要与各种括号搭配使用（括号、大括号、方括号、尖括号）。

（9）“...”符号

“...”符号用于数据外部初始化，表示相关数据需要从数据文件中查找。

（10）“..”符号

“..”符号用来表示区间范围，从前一个数值开始到后一个数值结束。

（11）“.”符号

“.”符号表示选取 tuple（元组）中的某部分。

（12）“|”符号

当元组变量间存在一定关系，且进行初始化时用“|”符号表示前面的元组类型成员来

自后面相对应的成员。

（13）in

in 在 forall、sum 等关键字中使用，如“forall (X in Y)表达式”表示对 Y 中的任意元素 X，执行表达式中相应的语句。

（14）“#”符号

“#”符号在数据初始化（参数赋值）中使用，用于将变量与相应的值进行对应。

（15）“:”符号

在约束条件中“:”符号可以给每个约束条件加标签，以标识各个约束。

（16）注释符号

OPL 建模语言具有两种注释方法。

① 第一种是：

```
/*注释内容*/
```

这种形式中注释内容是可以换行的，一般用于注释内容较多的情况，例如，在模型开头对这个模型进行简单介绍。

② 如果只是一行末尾的注释，可以是：

```
//注释内容
```

这种形式中注释内容是不能换行的，它一般用于标注某一代码行。

2.4 CPLEX OPL 数据类型

2.4.1 基本数据类型

OPL 中最常用的基本数据类型有整型数据、浮点型数据和字符型数据。

（1）整型数据

在 OPL 中，整型数据用 int 定义，可用于数据元素或决策变量的定义。OPL 还包含整数常量 maxint，它表示可用的最大正整数，该变量在 32 位电脑上为 2^{31}-1，在 64 位电脑上为 2^{64}-1。

int 的用法一般有两种形式：

```
int  变量名=数值;
int  数组变量名[对应的变量名]=[数值 1，数值 2，数值 n];
```

例如：

```
int a = 6;
```

用于定义值为 3 的整数 a。

整数变量初始化还可以通过表达式来指定。例如：

```
int n = 6;
int size = n*n;
```

用于定义整数变量 size 且对其赋值为 n 的二次方。

（2）浮点型数据

OPL 用 float 定义双精度浮点型数据。OPL 还有预定义的浮点常数 infinity，用于表示无

穷大数值常量。Infinity 在 CPLEX 中的具体值为 1e+20，如果问题中出现比该数值大的 float 值，则将该数值自动置为 1e+20。

浮点型数据有两种表现形式：

①十进制小数形式，由数字和小数点组成。

②指数形式，如 123E3 或者 123e3 都代表 $123x10^3$。注意 e 或者 E 前面必须有字，且后面的指数必须为整数，如 e3、2e3.5、.e3、e 都是无效的指数形式。

浮点型数据的定义方式类似于整数型数据。例如：

```
float f = 5.1;
```

用于定义值为 5.1 的浮点数 f。该浮点数的值可以通过任意表达式来指定。

（3）字符型数据

OPL 支持字符串数据类型。OPL 中的字符型数据是指由 string 定义的数据类型，其语法为：

```
{string)变量名= {string1, string2, string3,... ,string n};
```

例如：

```
{string} Tasks = {"masonry", "carpentry", "plumbing", "ceiling", "roofing",
"painting", "windows", "facade", "garden", "moving" };
//定义字符串，字符串名称为 Tasks
```

2.4.2 特殊数据类型

CPLEX OPL 中可以使用四种特殊类型的数据结构：区间、数组、元组和集合。

（1）区间

区间型数据使用 range 标识符来进行定义，用于定义一定区间范围的连续数据。整数型区间数据在数组、聚合运算、查询和限定条件中经常使用，所以在 OPL 中经常定义整数型区间数据，在少数情况下会定义实数型区间数据。

整数型区间数据定义语法为：

```
range 变量名=a..b;
```

其中：range 为区间型数据定义关键字，a 和 b 是两个整数且 b>a，a..b 表示区间[a，b]间的所有整数。

参数 a 和 b 可以直接给定数值，也可以给定表达式，例如：

```
range id= 4..7;//直接赋值区间上下限的方式
int n=4;
range id2 = n..n+3;//通过表达式确定区间上下限的方式
```

区间型数据定义之后，可以将其用作数组的索引。当 b<a 时，区间变量为空区间，OPL 自动将其转换为 0..-1。

整数区间型数据通常有如下三个用法：

（i）用于定义数组索引

```
range R=1..5;
```

```
int A[R];  //定义整数数组 A，其元素为：A[1]、A[2]、A[3]、A[4]和 A[5]
```

（ii）用于迭代循环

```
range R = 1..10;
forall(i in R){
  //循环操作，连续执行 10 次，执行时 i 值依次为 R 中的数据：1、2、……、10
  …
}
```

（iii）用于确定决策变量的取值区间

```
range R = 1..10;
dvar int i in R; //定义整数决策变量 i，i 的取值区间为[1,10]
```

实数型区间数据定义语法同整数型区间数据定义语法类似，不同的是 a 和 b 为实数（或称之为浮点型数据）。实数型区间数据定义了一个实数区间，一般用作限定决策变量的取值范围，以提高优化运算的速度。

例如：

```
range float R = 1.0..100.0; //定义浮点区间变量 R 为 1 到 100 之间的实数
dvar float x in X; //定义决策变量 x 的取值区间为 R
```

（2）数组

需要存储多个同类型数据时通常需要使用到数组，OPL 中可以定义一维数组和多维数组，且数组中可以存储两种基本类型数据：字符型数据和数值型数据，注意同一个数组中只能存储同一种类型的数据。

一维数组在 OPL 中是最简单的数组，定义语法为：

```
数据类型    数组名[区间]=数值列表;
```

数据类型根据数组中存储的数据类型可以是 int、float 和 string，数组名使用英文字符来给出，区间给出数组的长度和索引，数值列表给出该数组所存储的具体数值。

例如：

```
int a[1..4) = [10, 20, 30, 40];
float f[0..3] = [1.2, 2.3, 3.4, 4.0];
string s[1..3] = ["Monday", "Wednesday",”Friday”);
```

第一行定义长度为 4 的一维整数数组 a，该数组中的元素依次为 a[1]、a[2]、a[3]和 a[4]，各个元素存储的值依次为 10、20、30 和 40。第二行定义长度为 4 的一维实数数组 f，该数组中的元素依次为 f[0]、f[1]、f[2]和 f[3]，各个元素存储的值依次为 1.2、2.3、3.4 和 4.0。第三行定义长度为 2 的一维字符数组 s，该数组中的元素依次为 s[1]、s[2]和 s[3]，各个元素存储的值依次为“Monday”、“Wednesday”和“Friday”。

除了上述直接定义数组并赋值的方式之外，还可以先定义数组的区间，在定义数组并赋值，例如：

```
range R =1..4; //定义区间
int a[R] = [10, 20, 30, 40]; //定义长度为 4 的整数数组并赋值
```

除了使用整数区间型数据来定义数组的长度和索引之外，还可以使用字符串集合来定义数组的区间，例如：

```
{string} Days = {'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', "Saturday", 'Sunday'};
int startTime[Days]=[8, 8, 9, 9, 9, 10, 10];
```

此时要引用 startTime 中的数值，索引需要使用 Days 中的字符串，例如 startTime 中的第二个元素值需要使用 startTime[“Tuesday”]来实现。

OPL 支持多维数组的定义，定义语法一般形式为：

```
类型关键字 数组名[指标集 1][指标集 2]…[指标集 n]=[列表 1][列表 2]…[列表 n];
```

例如：

```
int a[1..2][1..3]=[[1,2,3], [4,5,6]];
```

使用整数型区间数据定义两行三列的二维数值 a 并对其进行赋值。

还可以组合不同类型数据作为二维数组的索引，例如：

```
{string} Days ={“Friday”,”Monday”}
int a[Days][1,.3] =[[1,2,3], [4,5,6]];
```

这两行代码也定义了一个两行三列的二维数组 a，在引用该数组中的元素时行索引使用”Friday”和”Monday”，例如引用 a 的第二行第三个数据使用 a[“Monday”][3]。

数组型数据必须先定义然后才能够使用，数组中的元素既可以作为独立元素来使用，也可以作为变量索引来使用，例如：

```
{string} Products = {"gas", "chloride"}; //定义字符型数组
{string} Components = {"nitrogen", "hydrogen", "chlorine"}; //定义字符型数据
float Demand [Products][Components] = [[1,3,0],[1,4,1]]; //定义数值型二维数组
float Profit[Products] = [30,40];//定义数值型一维数组
float Stock[Components] = [50,180,40]; //定义数值型一维数组
dvar float+ Production[Products]; //定义决策变量时引用字符型数组确定变量大小和索引
maximize sum(p in Products)   profit[p]*Production[p]; //目标函数使用 Products 中的 p 做索引
subject to{
  forall (c in Components) //数组中的元素作为循环变量
       sum(p in Products)
            Demand[p] [c]*Production[p] <= stock[c]; //数组元素作为约束条件中变量的索引
}
```

（3）元组

同一个数组中每个元素必须是同一类型的数据，但有时需要将不同类型的数据组合成一个有机的整体，以便于引用，该需求可以通过元组（tuple）来实现。元组是一种将类型不同但相互间具有紧密相关性的数据聚集在一起的数据类型。

元组型数据使用 tuple 来进行定义，语法为：

```
tuple 元组名称{成员列表};
```

其中：tuple 为数据类型关键字，元组名称为该元祖数据结构的名称，大括号{}内定义该元祖各成员数据类型和名称，成员的语法形式为：

```
类型名 成员名;
```

例如：

```
tuple Point {
  int x;
```

```
    int y;
};   //定义元组类型数据 Point，它包含两个整数型字段 x 和 y
```

定义元组类型数据之后就可以定义单个元组对象、元组数组和元组的元组，例如：

```
Point p = <2,3>; //定义单个元组对象，即一个点
Point point[i in 1..3] = <i, i+2>;
tuple Rectangle {
   Point lowLeftP;
   Point upperRightP;
}
```

示例中第二行定义长度为 3 的一维 Point 元组数组，并对数组中的每个对象赋值，等号左边的[i in 1..3]一方面指定该元祖数组长度为 3，另一方面指定 i 依次取值 1、2 和 3，结合等号右侧<i,i+2>实现对每个元祖中的 x 和 y 进行赋值，该行执行结束后获得元祖数组 point，其内容为[<1,3>,<2,4>,<3,5>]。

示例中后四行定义一个元祖 Rectangle，该元素包含两个元素 lowLeftP 和 upperRightP，且这两个元素为 Point 元素，具体到实际问题中，这四行定义了一个矩形框元祖，每个元祖包含左下角点的坐标和右上角点的坐标。

引用和访问元组中每个字段的方式是在元组对象名称中添加点和字段名，例如：

```
Point p = <2,3>;
int x1 = p.x;
```

这两行语句实现定义整数变量 x1 并使用元组对象 p 的字段 x 值对其进行赋值。

注意：元组中的字段不支持多维数组；元组中的字段可以设置关键字段，该字段可以作为元祖对象的唯一标识在模型中使用。

元组中字段支持的数据类型包括：

●基本数据类型（int、float 和 string）；

●元组（也称为子元组）；

●一维数值数组，即整数一维数组或实数一维数组；

●基本数据类型集合（集合见后续内容），例如整数型集合、实数型集合以及字符型集合。

元组中字段不支持的数据类型包括：

●元祖集合；

●字符串数组、元祖数组；

●任何数据类型的多维数组。

（4）集合

集合是由不重复元素组成的元素集合。OPL 支持使用任意类型数据的集合来进行建模，假设类型为 T，那么可以使用{T}或者 setof(T)来进行类型 T 集合的定义，例如：

```
{int} setInt = …; //定义一个整数型数据的集合
setof(Point) points= …;//定义一个 Point 元祖的集合
```

OPL 中可以对集合进行多种操作，操作关键词和含义如表 2-5，示例中两个集合为

S1={1,2,4}，S2={2,5,6}。

表 2-5 OPL 集合操作关键词和示例表

关键词	语法示例	示例结果
union	{int}u = S1 union S2（并集：将两集合元素合并）	u = {1,2,4,5,6}
inter	{int}i = S1 inter S2（交集：两集合中共同元素）	i={2}
diff	{int}d = S1 diff S2（差集：属于 S1 不属于 S2 的元素）	d={1,4}
symdiff	{int}sd = S1 symdiff S2（运行两集合并集和交集的差集）	sd={1,4,5,6}
first	first(S)（获得集合 S 的第一个元素）	first(S1)={1}
last	last(S)（获得集合 S 的最后一个元素）	last(S2)={6}
item	item(S,n)（获得集合 S 的第 n 个元素，下标从 0 开始）	item(S1,2)=4
ord	ord(S,item)（获得集合 S 中 item 的位置，下标从 0 开始）	ord(S1,2)=1
next	next(S,item)（获得集合 S 中 item 下一个元素）	next(S1,2)=4
prev	prev(S,item)（获得集合 S 中 item 前一个元素）	prev(S1,2)=1
nextc	nextc(S,item)（把集合元素连成环，调用 next）	nextc(S1,4)=1
prevc	prevc(S,item)（把集合元素连成环，调用 prev）	prevc(S2,2)=6

2.5 数据初始化赋值

2.5.1 数据初始化方式

OPL 模型中变量数据初始化主要有两种方式：内部初始化和外部初始化。内部初始化指将变量数据初始化过程放在变量定义的模型文件内。外部初始化指变量的数据放在变量定义的模型文件之外，一般数据放在独立的数据（.dat）文件中，或数据放在 Excel 表格中。

在模型（.mod）中内部初始化数据方式示例：

```
int b = 9;
int a[1..5] = [b+1, b+2, b+3, b+4, b+5]; //整数数组 a 初始化
```

数据外部初始化时，在模型文件中需要使用三个点…来定义该变量数据为外部数据，例如模型（.mod）文件中变量定义如下：

```
{string} Product ={"flour", "wheat", "sugar"};
(string) City = {"Providence", "Boston", "Mansfield"};
tuple Ship {
string orig;
string dest;
string P;
}
{Ship} shipData = …;
```

在另外的数据（.dat）文件中设置 shipData 需要初始化的数据，例如：

```
shipData =
{
<"Providence", "Boston", "wheat">
<"Providence", "Boston", "flour">
<"Providence", "Boston", "sugar">
<"Providence", "Boston", "wheat">
```

```
};
```

数据内部初始化和外部初始化具有如下的区别：

（1）符号的区别：内部初始化数据之间必须加逗号；外部初始化的.dat 文件中，数据之间的逗号是可选的，对于没有任何特殊字符的字符串，甚至连引号也是可选的。

（2）内存分配的区别：内部数据在模型中首次使用时将进行初始化，即 OPL 不会给未使用到的内部初始化数据分配任何内存。外部数据模型运行开始进行 dat 文件解析时进行初始化，并且无论对应变量是否被使用都会为其分配内存。

2.5.2 数组初始化

（1）内部初始化

直接在模型文件中对变量进行赋值的内部初始化方式前面有较多的示例，这里不再赘述，而内部初始化还有如下两种方式：指令初始化和预处理脚本初始化。

指令初始化过程指 OPL 支持定义数值时通过表达式指令来初始化数值，该方法可以简化数学模型的编程，例如：

```
int a[i in 1..10] = i+1;
```

该命令定义一个含 10 个元素的数组 a，并对其每个元素 a[i]初始化数值为 i+1。

指令初始化还可以对多维数组进行初始赋值，例如：

```
int b[i in 0..4][j in 0..9] = 10*i + 5*j;
```

该命令定义了 5 行 10 列的二维整数数组 b，并对每个元素 b[i][j]初始化数值为 10*i+5*j。

指令初始化还可以实现二维数组的转置，例如：

```
int b[Dim1] [Dim2] = … :
int c[j in Dim2][i in Dim1] ≡ b[i][j];
```

预处理脚本是指 mod 文件在模型主体建模代码之前使用 execute 的方式进行的数据处理脚本，该脚本语法规范类似于 javascript，因此可以在该脚本中进行比较复杂的数据处理和初始化。例如：

```
range Days = 1..9;
int a[Days];
execute {
   for(var i in Days){
      a[i]= 2*i;
   }
}
```

上述代码第 1 行定义一个[1,9]的区间变量 Days，第 2 行定义索引为 Days 的一维整数数组 a，第 3-7 行为预处理脚本，其中第 4-6 行循环对数组 a 进行初始化赋值，实现 a[1] = 1、a[2]=4、a[3]=9、…、a[9]=81 的初始化赋值。

（2）外部初始化

数组使用外部初始化的方式赋值需要在模型（.mod）文件中定义进行定义并在等号右侧设置三个…，如：

```
int a[1..2] [1..3] = ...;
```

```
{string} Days ={“Monday”,”Tuesday”,” Wednesday”,”Thursday”,”Friday”};
int b[Days] = …;
```

然后在数据（.dat）文件中设置数组的具体值。数组外部初始化也有两种方式：列出数值、指定索引和值对。为了对前述变量 a 和 b 进行赋值，可以在 dat 文件中设计如下：

```
a=[
    [10,20,30],
    [40,50,60]
];
b = #[
    "Monday": 1,
    "Tuesday": 2,
    "Wednesday”: 3,
    "Thursday": 4,
    "Friday": 5
]#;
```

二维数组 a 通过直接列出数值的方式进行初始化，一维数组 b 使用索引和值对的方式进行初始化。通过索引和值对进行初始化时，必须使用定界符#[和]#来代替[和]，且索引和值对顺序可以随意安排。

所以还可以在 dat 文件中使用下列方式对 a 和 b 进行初始化：

```
a=#[
    2:[40,50,60],
    1:[10,20,30]
]#;
b = [1 2 3 4 5];
```

2.5.3 元组初始化

（1）内部初始化

元组数据的初始化类似于数组，只是将数组初始化中的[]符号改为<>符号。例如直接在模型文件中内部初始化元组如下：

```
tuple Point
{
    int x;
    int y;
}//定义元组 Point
Point p1=#<y:1,x:2>#;
Point p2=<5,3>;
```

该模型中定义了两个 Point 类型的元组对象 p1 和 p2，其中：p1 的 x 和 y 值按照索引和值对的方式初始化值 2 和 1；p2 按照元组字段顺序给 x 和 y 赋值 5 和 3。

（2）外部初始化

下面示例显示元组中有一维数组字段时采用外部初始化的方式进行赋值，模型（.mod）文件先定义一个元组类型 Rectangle，然后定义一个外部初始化数据的元组对象 r，再使用脚

本将 r 的值输出出来：

```
tuple Rectangle {
    int id;
    int x[1..2];
    int y[1..2];
}
Rectangle r = …;
execute
{
  writeln(r);
}
```

数据（.dat）文件中定义 r 的数据如下：

```
r=<1, [0,0],[10,10]>;
```

2.5.4 集合初始化

集合数据初始化可以使用三种方式：内部初始化、外部初始化和泛型初始化。

（1）内部初始化

集合数据内部初始化可以通过在模型文件中列举所包含的数值进行初始化，或者使用集合的操作符进行初始化，例如：

```
{int} s1 = {1,2,3};
{int} s2 = {1,4,5};
{int} i = s1 inter s2;
{int} j = {1,4,8,10} inter s2;
{int} u = s1 union {5,7,9};
{int} d = s1 diff s2;
{int} sd = s1 symdiff {1,4,5};
```

前两行代码通过直接列举数值的方式为集合对象 S1 和 S2 赋值，后五行通过集合的操作符定义了五个集合变量并对其进行赋值，赋值结束后，i={1}，j={1,4}，u={1,2,3,5,7,9}，d={2,3}，sd={2,3,4,5}。

可以使用 asSet 使用区间表达式来进行集合数据的初始化，如下列命令实现将 1、2、…、10 这十个数字赋给集合 S1。

```
{int} s1 = asSet(1..10);
```

注意，如果直接将一个集合变量赋值给另一个集合变量，则更改其中一个集合变量的内容，另一个集合变量内容会随之改变，如果不希望两个集合变量之间值相关联，则需要使用泛型表达式的方式来进行内容的复制，具体看下面示例：

代码	{int} s1={1,2};
	{int} s2=s1;
	execute{
	  s2.add(3);
	  writeln(s1);
	}
		{int} s1={1,2};
		{int} s2={ i | i in s1};
		execute{
		  s2.add(3);
		  writeln(s1);
		}

结果	{1 2 3}	{1 2}

（2）外部初始化

通常都是使用外部初始化方式对集合元素进行初始化，与数组和元组等数据外部初始化过程类似，在模型（.mod）文件中定义集合变量，加上=…表示该集合数据由外部数据文件赋值，然后再在数据（.dat）文件中列出集合变量的数据。例如模型文件内容如下：

```
tuple Precedence {
   int before;
   int after;
}
{Precedence} precedences = ...;
{string} days=…;
```

数据（.dat）文件中进行数据初始化如下，其中第一行初始化使用三个元组对象 Precedence 的集合，每个元素对象中 before 和 after 字段的值依次由<>指定，第二行直接列出三个字符串初始化 days 集合变量。

```
precedences={<3,4>,<6,5>,<2,4>};
days ={"Monday","Tuesday","Friday"};
```

（3）泛型初始化

CPLEX OPL 可以在模型中使用一种通用泛型表达式进行集合变量数据初始化。

例如：

```
{int} s={i | i in 1..10: i mod 3 ==1};
```

该命令使用中间变量 i 遍历 1 到 10 之间的数据，如果数据 i 满足同 3 取余为 1，则将数据 i 赋给集合 s，所以 s={1,4,7,10}。

例如：

```
{int} a[e in 3..4]={i | i in 1..10: i mod e ==0};
```

该命令中，等式左侧定义长度为 2 的整数集合数组 a，该集合有两个索引 3 和 4，即集合 a 包含集合元素 a[3]和 a[4]；等式右侧提取满足 i in 1..10: i mod e ==0 的数值 i 赋给对应的集合元素，从而实现对 a[3]初始化赋值{3,6,9}，对 a[4]初始化赋值{4,8}。

2.6 OPL 与 Excel 文件数据交互

当模型中涉及的参数数据量较大时，直接在数据（.dat）文件中编辑容易出现错误，而且难以阅读，所以会将格式化的数据存储在 Excel 表格文件中，模型运算之前通过初始化将 Excel 数据表中的数据读出来并赋值给对应变量或数组等。当模型优化运算出结果之后，同样可以将数据直接存入 Excel 数据表，便于后续查看和进一步分析。注意，OPL 模型读取 Excel 数据或将变量写入 Excel 文件的交互过程都是在数据（.dat）文件中编程实现的。

2.6.1 OPL 与 Excel 文件的连接

CPLEX 连接 Excel 文件需要用到命令为 SheetConnection，具体语法为：

```
SheetConnection sheetId( "excelFileName");
```

其中：SheetConnection 为连接 Excel 文件的命令；sheetId 为当前链接所起的连接名称，

当一个模型中连接多个 Excel 文件时，通过给不同 Excel 文件设置不同的 sheetId 名称可以进行连接标识；excelFileName 为 Excel 文件名称，该名称需要包含文件后缀。

例如 CPLEX 自带项目 oil 中的 oil.dat 数据文件中使用下列语句连接 Excel 文件：

```
SheetConnection sheet("oilSheet.xls");
```

在该链接中仅给出 Excel 文件的完整名称"oilSheet.xls"，并没有给出该文件的完整路径，所以该文件一般放在项目模型文件和数据文件的同一个文件夹中。

2.6.2 从 Excel 文件中读取数据

使用命令 SheetRead 可以将 Excel 文件中的数据读取出来并对 OPL 模型中的相关数据变量进行初始化，该命令语法如下：

```
a from SheetRead (sheetId, "'sheetName'!rangeName") ;
```

其中，a 为模型文件中定义的数据变量名称；from 为读取数据的关键字；SheetRead 为命令函数；sheetName'!rangeName 分别指定数据所处的工作表名称和数据区域名称。

例如 CPLEX 自带项目 oil 中的 oil.dat 数据文件中使用的读取数据语句：

```
SheetConnection sheet("oilSheet.xls");
Gasolines from SheetRead(sheet,"'gas data'!A2:A4");
Oils from SheetRead(sheet,"'oil data'!A2:A4");
Gas from SheetRead(sheet,"'gas data'!B2:E4");
Oil from SheetRead(sheet,"'oil data'!B2:E4");
```

第 2 和 4 行分别将数据表“oilSheet.xls”中工作表“gas data”中的数据区域“A2:A4”和“B2:E4”的值赋给一维数组 Gasolines 和二维数组 Gas，第 3 和 5 行分别将数据表“oilSheet.xls”中工作表“oil data”中的数据区域“A2:A4”和“B2:E4”的值赋给一维数组 Oils 和二维数组 Oil。

OPL 以只读方式打开 Excel 表格并从中读取数据，支持读取的数据类型包括：

（1）整数、浮点数、字符串或元组的集合。

（2）整数、浮点数或字符串单个数值。

（3）整数或浮点数的数组、一维/二维字符串或一维元组数据的数组。

OPL 能够识别 Excel 数据区域的名称。Excel 数据区域的名称是用于表示一个单元格、单元格范围、公式或常量值的一个字符或一串字符，使用名称表示数据区域更易于理解对应区域数据的特点，例如使用 Nutrients 来引用表格中 B4:J15 范围的数据要比直接使用 B4:J15 更易于理解。

当 Excel 特定数据区域定义了名称，则可以通过如下方式读取该区域数据并将其赋值给相关数据变量，例如下列命令：

```
SheetConnection sheetData("C:\\example\\production.xls");
prods from SheetRead (sheetData, "Product");
```

第一行是建立绝对路径下 production.xls 文件的连接并命名为为 sheetData，第二行使用 SheetRead 命令读取该连接中命名为 Product 的区域数据初始化变量 prods。

2.6.3 写入 Excel 表格

优化过程的结尾，需要存储优化的结果，可以将结果存储在 Excel 表格中，需要用到的命令关键字为 SheetWrite，该命令同 SheetRead 格式类似。

将模型数据写入 Excel 的语法：

```
a to SheetWrite{sheetId, "sheetName!rangeName"};
```

其中：a 为模型中的变量，一般为决策变量，也可以是普通的统计变量；to 为关键字；SheetWrite 为写入命令，其后括号中的两个参数依次为连接名称和数据区域名称。

例如 CPLEX 自带项目 oil 中的 oil.dat 数据文件中最后两行语句：

```
a to SheetWrite(sheet,"RESULT!A2:A4");
Blend to SheetWrite (sheet,"RESULT!B2:D4");
```

第一行语句将求解的决策变量 a 值写入连接标识为 sheet 的 Excel 文件中的 RESULT 工作表的 A2:A4 区域；第二行语句将求解的决策变量 Blend 值写入连接标识为 sheet 的 Excel 文件中的 RESULT 工作表的 B2:D4 区域。

2.7 优化模型设计

在 CPLEX 中设计优化模型主要包括：

- 问题参数定义和数据初始化；
- 决策变量定义；
- 目标函数设计；
- 约束条件设计。

其中问题参数设计和数据初始化参考前述数据类型和数据初始化赋值两节内容进行设计即可，本节主要介绍后面三项内容。

2.7.1 决策变量定义

决策变量是模型需要求解的未知量，这些变量一般具有特定的取值区间，该区间限定这些变量取值的空间。

OPL 在模型中使用关键字 dvar 来定义决策变量，通用语法如下：

```
dvar  数据类型标识符   变量名称   in  数据区间;
```

其中：

- dvar 为决策变量定义关键字；
- 数据类型可以使用整型（int）、浮点型（float）和布尔型（boolean）中的一个标识符限定，当数据类型标识符 float、int 后面跟“+”或者“-”号时分别表示该决策变量为大于等于 0 的值或小于等于 0 的值；当数据类型标识符为 boolean 时，表示决策变量为布尔型变量（0-1 变量）；
- 变量名称设定符合命名规范的决策变量名称和数量；
- “in 数据区间”不是必需的，当不指定该项时表示变量取值区间由数据类型标识符限定。

例如：

```
dvar int+ x in 0..100;
dvar int y[1..5] [1..10] in 0..100;
dvar float- z;
dvar boolean w[0..7];
```

第一行命令定义一个整数决策变量 x，该变量的取值区间为 0 到 100；第二行命令定义了一个 5 行 10 列的二维整数决策变量数组 y，该决策变量数组中的任何一个值的取值范围都在[0,100]区间内；第三行定义一个浮点数决策变量 z，该变量取值区间为负无穷大到 0；第四行定义一个一维 0-1 决策变量数组 w。

OPL 还支持决策表达式，即能够使用决策变量（dexpr）的表达式。OPL 中提供了特定语法将元素动态地收集到数组中。

关键字 dexpr 允许创建可复用决策表达式。如果表达式在原始问题方面有特定的含义，那么在预处理阶段将其编写为决策表达式（dexpr）会提高模型的可读性。

语法：

```
"dexpr"  数据类型关键字  决策变量名"="  表达式
```

例如，scalableWarehouse.mod 示例将固定总成本表示为决策表达式：

```
dexpr int TotalFixedCost = sum( w in Warehouses ) Fixed * Open[w];
dexpr float TotalSupplyCost = sum (win Warehouses, s in Stores) SupplyCost[s][w]* Supply[s][w];
```

这样，在 OPLIDE 中显示了定义的两个总成本表达式及其值。还可以使用决策表达式数组。例如：

```
dexpr int slack[i in r] = x[i] - y[i];
```

由于仅保留了“定义”，因此可以高效地处理该数组。

2.7.2 目标函数设计

运筹优化模型的目标函数或者为最大化或者为最小化，CPLEX 中使用 maximize 和 minimize 两个关键字来设计目标函数，设计语法如下：

```
maximize/minimize  表达式;
```

例：

```
maximize    sum(p in Products) Profit[p]*Production[p];
```

该行代码设计优化目标为总利润最大的目标函数，从右侧表达式中可以推测问题的决策变量为生产各种产品的产量 Production[Products]，其中：Profit[p]*Production[p]为第 p 中产品生产量与该产品单件利润的乘积；sum(p in Products)是对 Products 中的每种产品利润求和，即生产全部产品的总利润；maximize 设定优化求解目标为总利润最大。

例：

```
minimize sum(p in Products, o, d in Cities) Cost[p] [o] [d] *Trans [p] [o] [d] ;
```

该行代码设计优化目标为总运输成本最小的目标函数，从右侧表达式中可以推测问题的决策变量为城市之间运输的各种产品的数量 Trans[Products][Cities][Cities]，其中：Cost[p] [o] [d] *Trans [p] [o] [d]为从城市 o 向城市 d 运送的第 p 种产品的单位成本与运输量的乘积，即城市

间单种产品的运输成本；sum(p in Products, o, d in Cities)是对 Products 中的每种产品从 Cities 中各个城市之间运输的成本求和，即全部产品在全部城市间的运输总成本；miminize 设定优化求解目标为运输总成本最小。

2.7.3 约束条件设计

运筹优化问题的约束条件为一组同决策变量相关的布尔表达式，可以使用 constraints 或 subject to 块来进行设计。

例如：

```
minimize sum(p in Products) (insideCost [p] *inside[p] + outsideCost [p] *outside [p]);
subject to {
   forall (r in Resources) sum(p in Products) consumption[p,r] * inside[p] <= capacity[r]; //资源约束
   forall (p in Products) inside[p] + outside[p] >= demand[p]; //需求约束
}
```

该示例为 CPLEX 自带示例项目 production 中“production.mod”模型的一部分，该模型描述某公司需要满足市场对三种产品特定数量的需求，这三种产品可以自制也可以外包，自制成本要比外包成本低，自制每种产品需要消耗一定数量的两类资源且这两种资源数量有限。优化决策目标是确定各种产品自制数量和外包数量以实现总成本最低。

示例代码中第一行是目标函数，该目标函数中的 inside[p]和 outside[p]分别为产品 p 的自制和外包数量决策变量。这些决策变量需要满足两个约束条件：（1）第三行自制产品消耗的每种资源总量不能超过资源的拥有量约束；（4）第四行自制和外包数量之和不能小于需求量约束。

约束条件 subject to 块中的 forall 相当于数学中的∀符号，即对括号内任何值都需要满足后面的条件表达式； sum 相当于数学中的Σ符号，即对括号类任何值进行后续表达式的累计求和。

有些问题的约束条件会根据一些变量取值不同而呈现不同的表达式，这是就需要使用 if-else 分支结构来实现了，例如：

```
if (d>1){
   abs (freq[f] - freq[g]) >= d;
}else {
   freq[f] == freq[g];
}
```

这种条件约束表达式中不能包含决策变量，也不能使用 forall 语句。如果约束条件存在这两种情况，可以考虑使用过滤条件的方式来实现。

例如：

```
    forall( n in Nurses , d in Weekdays : Vacations[n][d] == 1 )
        sum(s in Shifts : s.day == d) NurseAssignments[n][s] == 0;
```

第一行中是在 forall 关键字中对 n 和 d 进行过滤，第二行是在 sum 中对 s 进行过滤。过滤的基本语法是：

```
a in b: condition
```

其中：b 为区间变量或集合变量，condition 为条件表达式，该过滤条件实现将 b 中满足条件 condition 的 a 逐个提取出来进行操作。

过滤条件还可以在目标函数或脚本中使用，例如：

```
// The cities where we are doing business
{string} cities={"Paris","Berlin","Washington","Rio"};
{string} EuropeanMainCapitals = {"London","Paris","Berlin","Madrid","Roma"};
// Should we expand business in this city ?
dvar boolean x[cities];
// We want to expand business in Europe
maximize sum(c in cities: c in EuropeanMainCapitals) x[c];
subject to
{
    // We can expand business in 2 cities
    maxCityConstraint:
    sum(c in cities) x[c]<=2;
}
{string} expanded_cities = {c | c in cities : x[c]==1};
execute
{
  for(c in expanded_cities) writeln("We should expand business in ",c);
}
```

该示例优化问题是需要再欧洲一些城市开展业务，约束条件是最多只能在两个城市开展新业务，目标函数是开展业务的城市尽量在欧洲主要国家的首都。示例中给出两个字符型集合变量 cities 和 EuropeanMainCapitals，在目标函数设计中增加了过滤条件 c 必须是欧洲主要首都集合中的值，最后输出确定开展业务的城市，所有在倒数第四行对全部城市进行过滤并定义一个新字符集合。

结果输出为：

```
We should expand business in Paris
We should expand business in Berlin
```

在示例的约束条件上还有一行字符“maxCityConstraint:”，该字符串为约束条件标签，用于识别该约束条件。当模型约束条件比较多时，给每个约束条件进行标签标注有助于识别和分析运算的结果，例如通过约束标签可以从 IDE 问题窗口查看模型中约束的具体情况，可以比较容易地识别各个约束的松弛值和对偶值。约束标签的使用方式就是在对应约束前写上具有含义的字符并以冒号“：”结束即可。

2.8 形参

形参是 OPL 中最基本的元素，可用于聚合运算符、泛型表达式和 forall 语句等。

2.8.1 形参基本形式

最简单的形参具有以下形式：

```
p in S
```

其中，p 是形参变量，S 是 p 从中获取其值的集合。集合 S 可以是整数区间变、字符串集合、元组集合。

例如：

```
int n=6;
int s = sum(i in 1..n) i*i;
{string} Products = {"car", "truck"};
float cost[Products] = [12000,10000];
float maxCost = max(p in Products) cost[p];
{string) Cities = {"Paris", "London", "Berlin"};
tuple Connection
{
   string orig;
   string dest;
}
{Connection} connections = { <"Paris", "Berlin">,<"Paris", "London">};
float cost[connections) = [1000，2000];
float maxCost= max(r in connections) cost[r];
```

第二行求和运算符 sum 后形参中的集合为整数区间 1..6，第五行求最大值运算符 max 后形参中的集合 Products 为字符串集合，最后一行求最大值运算符 max 后形参中的集合 connections 为元组集合。

2.8.2 过滤条件的形参形式

如果需要使用条件来过滤形参范围，形参形式如下：

```
p in S：condition
```

根据将 S 满足过滤条件 condition 的所有元素分配给 p，例如：

```
int n=8;
dvar int a[1..n][1..n];
subject to
{
   forall(i,j in 1..8:i<j)    a[i][j] >= 0;
}
```

第二行定义了 8 行 8 列的整数决策变量 a，约束中的条件形参实现所有 i<j 时的决策变量 a[i][j]必须不小于 0 的条件约束，至于 i≥j 时的决策变量 a[i][j]不受不小于 0 的约束。

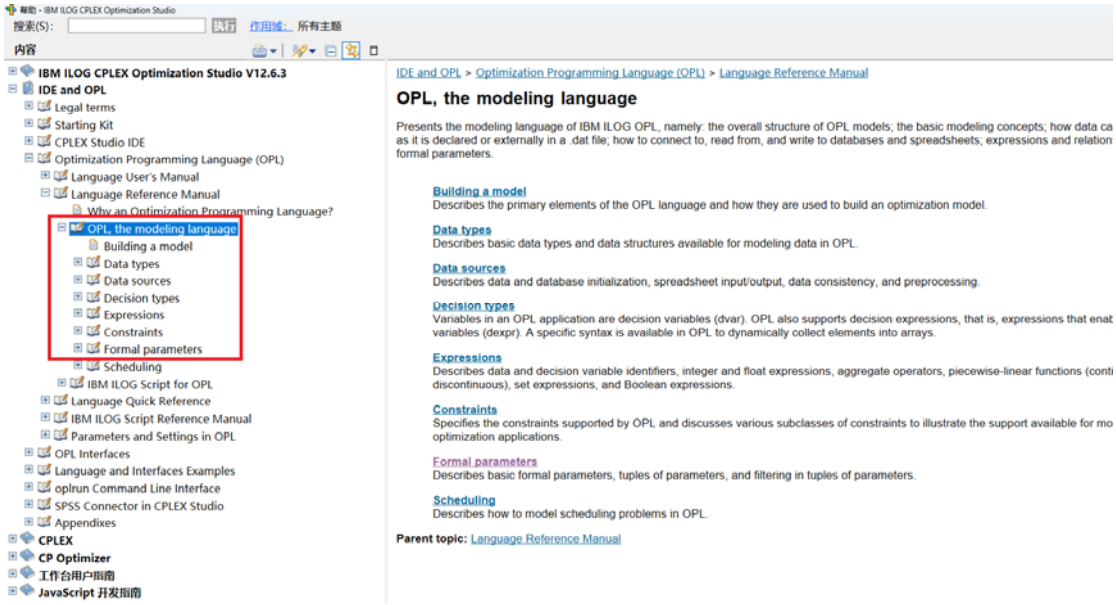
2.9 本章小结

本章主要介绍 CPLEX 软件建模基础知识，为后续进行车间调度问题数学模型的 CPLEX 优化程序建模奠定基础。本章首先介绍 CPLEX 发展历程和安装方式，然后对 CPLEX 项目以及运算符进行介绍和示例说明，同时对使用 CPLEX 构建运筹优化模型涉及的其他设置项目进行介绍和示例说明。在理解了本章关于 CPLEX 的基础知识之后，将可以比较容易的理解后续章节中关于各类车间调度问题 CPLEX 优化程序。

思考题

1. 在 CPLEX 中分别导入示例项目 oil、production 和 transp 项目，查看各个项目文件结构，理解模型语句，运行并查看分析优化结果。

2. 查看学习 CPLEX 软件的 help 系统中的“OPL, the modeling language”中的内容，并对照教程进行相关内容的练习。



3. 将 2.2 节中的示例项目问题使用单个模型文件编程实现，即数据采用内部初始化的方式实现，并进行优化运算。

4. 有优化问题的数学模型如下：

$$\min \quad z = 280\left(x_{11}+x_{21}+x_{31}+x_{41}\right)+450\left(x_{12}+x_{22}+x_{32}\right)+600\left(x_{13}+x_{23}\right)+730x_{14}$$

$$\text{st} \quad \begin{cases} x_{11}+x_{12}+x_{13}+x_{14} \geq 15 \\ x_{12}+x_{13}+x_{14}+x_{21}+x_{22}+x_{23} \geq 10 \\ x_{13}+x_{14}+x_{22}+x_{23}+x_{31}+x_{32} \geq 20 \\ x_{14}+x_{23}+x_{32}+x_{41} \geq 12 \\ x_{ij} \in I^{+} \quad (i=1\cdots4,\ j=1\cdots4) \end{cases}$$

使用 CPLEX 构建该优化问题的优化项目并求解。

5. 某糖果厂用原料 A、B、C 加工成三种不同牌号的糖果甲、乙、丙。已知各种牌号糖果中 A、B、C 含量、原料成本、各种原料每月限制用量、三种牌号糖果的单位加工费及售价如下表表示，试建立该厂每月利润最大的生产计划数学模型，并使用 CPLEX 建模和求解。

原料	甲	乙	丙	原料成本(元/kg)	每月限制用量(kg)
A	≥60%	≥20%		2.6	2000
B				1.8	2800
C	≤20%	≤50%	≤60%	1.2	1300
加工费(元/kg)	0.6	0.5	0.4		

售价(元/kg)	3.8	2.9	2.3		