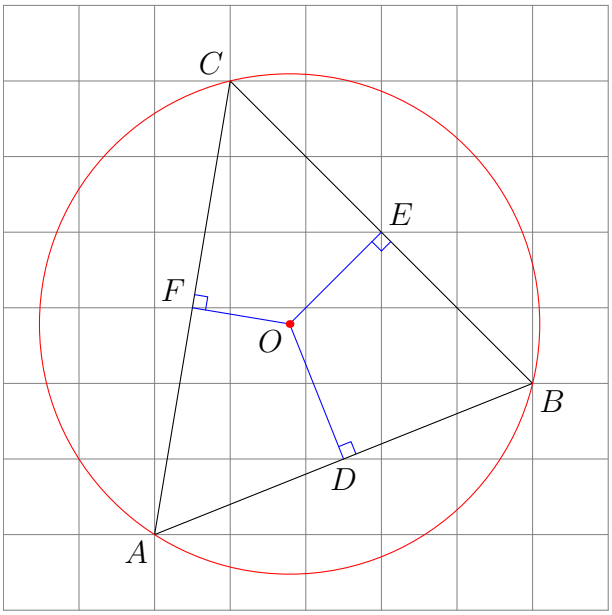
第三章　一个实例

本章将通过一个例子来使你对用 Asymptote 语言编程有一个初步的印象, 并对 Asymptote 的语法有一个大体上的了解. 通过本章的学习, 你将能够使用 Asymptote 编写出简单的程序, 作出不太复杂的图形. 后续各章将给出 Asymptote 语言的细节.

3.1　作图任务

假设现在你是一个初中几何教师, 正准备讲“三角形外接圆”这一内容. 在准备教案和课件的时候, 你觉得应该画一幅图来帮助你. 于是你边在纸上画草图, 边在心中确定了整个作图方案, 它看起来就像图 3.1 的样子.



$\triangle ABC$ 的外接圆 $\odot O$.
三角形的外心是三角形三条边的垂直平分线的交点.

图 3.1: 最终的作图计划.

在这幅图中需要分别作出几种不同的图形元素. 大体来说, 图中包括了一个背景网格、一个三角形、一个外接圆、几条中垂线、和一些文字.

3.2　背景网格

我们这里的背景网格是由一些横线和竖线组成. 每一条线可以由其两个端点来确定. 因此, 只要给定了两个点的坐标, 就可以画出一条线段了.

在 Asymptote 中, 点的坐标用 (x,y) 来表示, 其中 x 和 y 是实数, 分别表示该点的横坐标和纵坐标. 这种表示方法跟我们的习惯是相符的 (实际上, 在 Asymptote 的内部点的坐标是用复数表示的, 见第 4.4 节). 可以把点的坐标赋值给一个变量, 它的数据类型是对型 (pair), 例如:

```
pair z = (1cm,5cm);
```

声明一个变量 z 并把它的值初始化为 (1cm,5cm), 在以后就可以用 z 来引用这个点了.

> 现在你可能会有一个疑惑, 我们在图 3.1 中并没有指出每个点的坐标分别是多少, 那么, "我应该把这个直角坐标系的坐标原点建在哪里呢?" 其实这并不重要, 随便你建在哪里.

现在我们不妨把坐标原点建在背景网格的左下角. Asymptote 认为, 横轴的正方向是水平向右的, 而纵轴的正方向是竖直向上的. 这也恰好符合我们的习惯.

把两个点的坐标用两个减号 -- 连接起来就是一条线段. 用 Asymptote 的术语来说, 线段是一种称为路径 (path) 的数据类型. Asymptote 提供了一个函数 draw 来画路径. 例如, 要画一条端点坐标分别为 (3cm,1cm) 和 (4.5cm,3.2cm) 的线段, 可以用命令:

```
draw((3cm,1cm)--(4.5cm,3.2cm));
```

请注意行尾的分号. 你可以在交互模式下试试看.

这里出现了一些诸如 8cm 之类的记号来表示 8 厘米的长度. 前面说过, 在一个点的坐标 (x,y) 中, x 和 y 是实数. 那么 8cm 是实数吗? 答案是肯定的, 你可以在交互模式下用下面的命令来验证这一点:

```
write(8cm);
```

这里的 write 是 Asymptote 提供的一个函数, 用来向终端屏幕或者文件输出数值等信息. 执行这个命令你将真的看到一个实数 226.771207143293. 它的意思是说, 在 Asymptote 看来, 8cm 就等于 226.771207143293.

> 有点迷惑吗? 其实很简单, Asymptote 预先定义了一些变量, 其中包括 cm. 而按照 Asymptote 的语法规则, 当一个数字和一个变量相乘并且数字在前时, 可以省略乘号. 因此 8cm 实际上就是 8 * cm. 这个规定可以使我们的程序看起来更容易一些 (详见第 4.5 节, 第 29 页).
> 当用实数来表示长度或者距离时, Asymptote 使用 PostScript 的"大点" (big point, bp) 作单位, 一个 bp 等于 1/72 英寸.

好了, 我猜你可能已经知道该怎样画背景网格了. 没错, 象这样就行:

```
draw((0cm,0cm)--(8cm,0cm));
draw((0cm,1cm)--(8cm,1cm));
draw((0cm,2cm)--(8cm,2cm));
......
draw((0cm,0cm)--(0cm,8cm));
draw((1cm,0cm)--(1cm,8cm));
draw((2cm,0cm)--(2cm,8cm));
......
```

这里我们总共有 18 条线要画, 你需要写 18 行 draw, 还好, 任务还不算很重. 但如果要画一个更大的网格怎么办?

Asymptote 中有类似于 C/C++ 中的循环结构来帮助我们做这种重复性工作. 请在交互模式下尝试:

```
for (int i = 0; i <= 2; ++i) write(i);
```

你会得到如下的输出结果:

```
0
1
2
```

这个例子演示了 `for` 循环结构. 我们首先定义了一个整型循环变量 `i`, 并将它的值初始化为 0; 然后设定循环条件 `i <= 2`, 即只要这个条件仍然满足, 就重复执行循环体; 接下来的 `++i` 是说每次执行完循环体后, 将变量 `i` 的值增加 1. 本例的循环体只有一句话, 就是 `write(i);`, 在终端屏幕上显示变量 `i` 的值.

下面是我们使用循环结构来画背景网格的程序清单:

```
// 背景网格
for (int i = 0; i <= 8; ++i) {
  real x = i * cm;
  // 横线
  draw((0,x)--(8cm,x));
  // 竖线
  draw((x,0)--(x,8cm));
}
```

你发现一些行是由 `//` 开头的, 它们是*注释*. 当 `Asymptote` 读到 `//` 时, 它会忽略从这个地方开始直到这一行结束的全部内容. 在程序中写注释的目的是为了让人更容易读.

这个程序仅仅包含了一个循环结构. 花括号 `{}` 包围的部分是循环体. 由于循环体多于一条语句, 因此花括号是必需的. 循环体中定义了一个实型 (`real`) 变量 `x`, 后面调用了两次 `draw` 函数来分别画横线和竖线. 程序的执行结果见图 3.2.

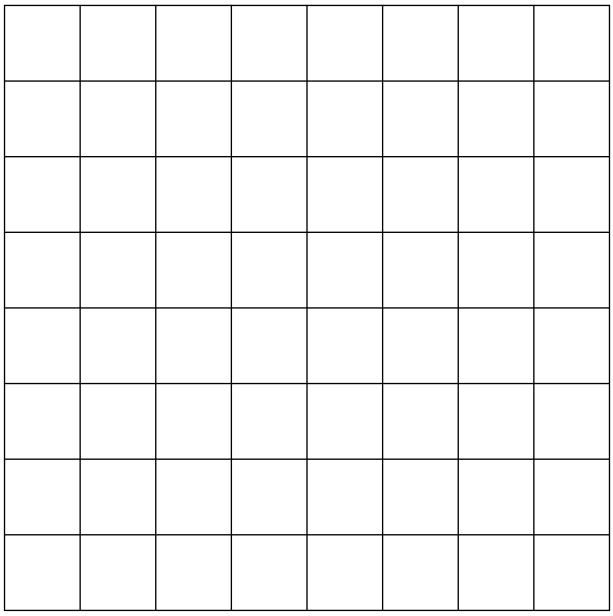


图 3.2: 背景网格的初步实现.

比较一下图 3.2 和我们的最终目标图 3.1 就会发现不同. 由于网格是属于背景性质的, 它看起来不应该很醒目. 因此我们希望用较细的线条和较浅的颜色来画它.

Asymptote 的数据类型中有一种叫做笔 (pen). 用这种数据可以定义所画线条的颜色、粗细、线型等性质. 在默认的情况下, Asymptote 用黑色、0.5bp、实线来画线条. 下面我们把程序作一些修改, 来用灰色和 0.2bp 的笔画背景网格. 为此我们定义一个 pen 类型的变量 helpline, 并在两个 draw 函数中使用它. 现在我们的程序清单为 (输出结果见图 3.3) :

```
// 辅助线
pen helpline = linewidth(0.2bp) + gray(0.5);

// 背景网格
for (int i = 0; i <= 8; ++i) {
  real x = i * cm;
  // 横线
  draw((0,x)--(8cm,x), helpline);
  // 竖线
  draw((x,0)--(x,8cm), helpline);
}
```

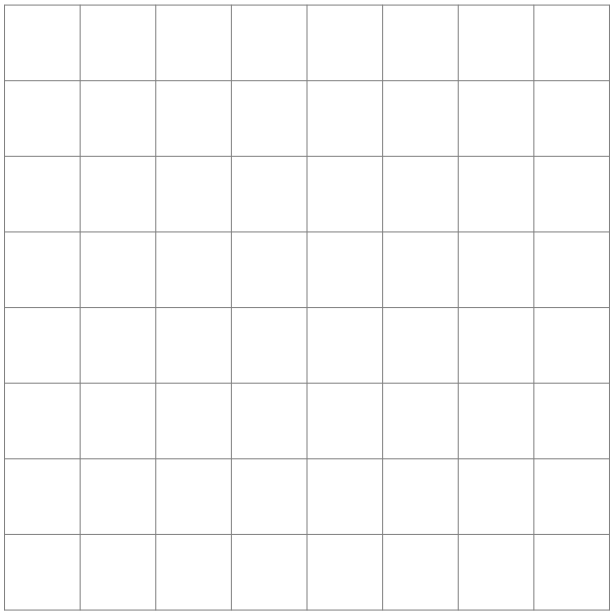


图 3.3: 修改过的背景网格.

3.3 三角形

我们把三角形 ABC 的三个顶点分别放在 (2cm,1cm), (7cm,3cm), 和 (3cm, 7cm) 处, 见图 3.1.

我们没必要一段一段地画出三角形的三条边, 而可以一次就完整地画出整个三角形的封闭路径. 程序如下 (作图结果见图 3.4) :

```
1  // 辅助线
2  pen helpline = linewidth(0.2bp) + gray(0.5);
3
4  // 背景网格
5  for (int i = 0; i <= 8; ++i) {
6    real x = i * cm;
7    // 横线
8    draw((0,x)--(8cm,x), helpline);
9    // 竖线
10   draw((x,0)--(x,8cm), helpline);
11 }
12
13 // 三角形 ABC
14 pair a = (2cm,1cm), b = (7cm,3cm), c = (3cm, 7cm);
15 draw(a--b--c--cycle);
```

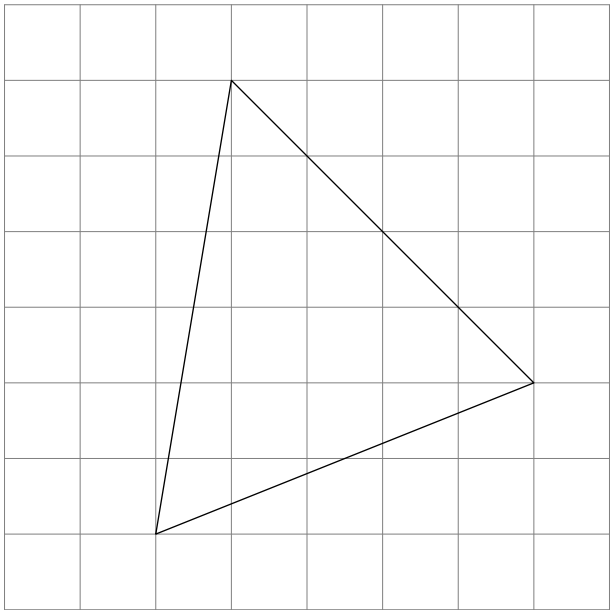


图 3.4: 在背景网格上添加三角形.

在程序的第 14 行, 我们定义了三个 `pair` 类型的变量用来记录三角形的三个顶点坐标; 第 15 行则直接调用 `draw` 函数用默认的笔画出了整个三角形. 注意, 三角形的路径是 `a--b--c--cycle`, 这里最后的 `cycle` 将把终点 `c` 和起点 `a` 连接起来形成一个封闭的路径. 封闭的路径和开放的路径不同, 差别之一是, 前者可以用 `fill` 函数填充颜色而后者不行. 注意, `a--b--c--a` 不是封闭路径, 虽然它的终点和起点是同一点.

路径不见得都是由直线段组成, 它可以是曲线. 请在交互模式下试验如下命令的结果并比较封闭和开放路径的区别:

```
draw((0,0)--(4cm,0)--(5cm,1cm)--(4cm,2cm)--(0,0));
```

```
draw((0,0)..(4cm,0)..(5cm,1cm)..(4cm,2cm)..(0,0), red);
draw((0,4cm)--(4cm,4cm)--(5cm,5cm)--(4cm,6cm)--cycle);
draw((0,4cm)..(4cm,4cm)..(5cm,5cm)..(4cm,6cm)..cycle, red);
```

3.4 外接圆

由于圆既常见又常用, Asymptote 专门提供了一个函数来产生圆形路径, circle(c, r), 其中 c 是 pair 类型, 表示圆心的坐标, r 是实数, 表示半径. 因此, 只要我们找到圆心和半径就可以用它来画外接圆了.

我们知道, 三角形的外心是三条边的垂直平分线的交点. 所以, 我们必须有至少两条垂直平分线, 并计算它们的交点. 计算两条直线的交点通常意味着要求解一个二元一次方程组, 不过不要担心, Asymptote 已经为我们准备好了这样一个函数, extension.

Asymptote 的内核其实只提供了一些基本的功能, 而大量的实用功能都能用这些基本的功能在 Asymptote 语言下实现. 这样, 人们可以把一些具有相关的通用功能的 Asymptote 代码组织到一起, 称为一个模块. 通过这种方式, 任何人都可以对 Asymptote 加以扩展. 几乎任何计算机程序设计语言都有类似的扩展方式, 例如 C/C++ 的库以及 TEX/LATEX 的宏包等. 在 Asymptote 的发行中提供了一些包括科技作图和 3 维作图等功能的基本模块.

计算两条直线交点的函数 extension 就定义在 math 模块中. 这个函数有 4 个参数, 前两个是一条直线上的两个点, 后两个是另一条直线上的两个点, 函数的返回值就是它们的交点.

现在, 我们需要确定一条边（例如 AB 边）的中垂线上的两个点. 显然其中之一可以取为 AB 的中点 D, 它的坐标为 0.5(a + b)（根据前面讲过的规则, 我们省略了一个乘号）. 而另外一点则可通过把 B 点绕 D 逆时针旋转 90° 得到. Asymptote 有一种称为变换 (transform) 的数据类型专门做这种平移、旋转、缩放等变换工作, 例如函数 rotate 就可以用来把一个点或者路径等图形元素绕某一特定点旋转一定的角度.

至此, 我们的程序清单如下（结果如图 3.5 所示）:

```
1  import math;
2
3  // 辅助线
4  pen helpline = linewidth(0.2bp) + gray(0.5);
5
6  // 背景网格
7  for (int i = 0; i <= 8; ++i) {
8    real x = i * cm;
9    // 横线
10   draw((0,x)--(8cm,x), helpline);
11   // 竖线
12   draw((x,0)--(x,8cm), helpline);
13 }
14
15 // 三角形 ABC
```

```
16 pair a = (2cm,1cm), b = (7cm,3cm), c = (3cm, 7cm);
17 draw(a--b--c--cycle);
18
19 // 中垂线、外心、外接圆
20 pair d = 0.5(a + b), e = 0.5(b + c), f = 0.5(c + a);
21 pair o = extension(d, rotate(90, d) * b, e, rotate(90, e) * b);
22
23 draw(circle(o, abs(o - a)), red);
24
25 draw(o--d, blue);
26 draw(o--e, blue);
27 draw(o--f, blue);
28
29 dot(o, red);
```

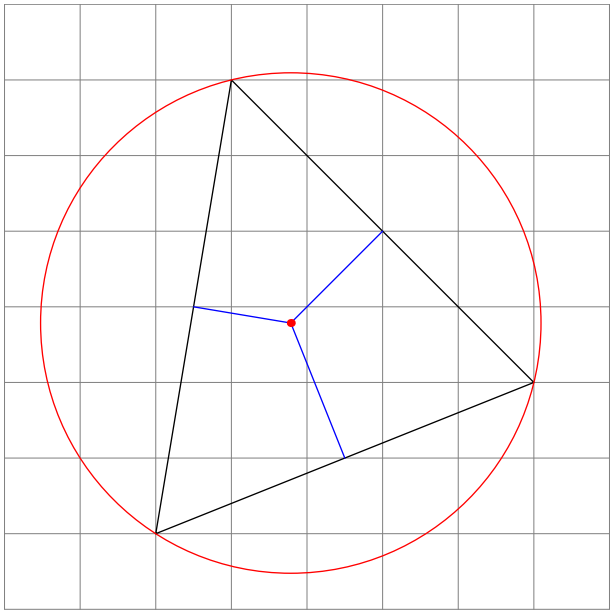


图 3.5: 添加了三条边的中垂线、外心、和外接圆.

第 1 行的 `import math;` 装入 `math` 模块, 我们要用到其中的 `extension` 函数. 第 20 行计算三条边的中点. 第 21 行计算 AB 和 BC 边中垂线的交点, 即三角形的外心 O, 其中 `rotate(90, d) * b` 得到“将 B 点绕 D 点逆时针旋转 90° 后的坐标”. 第 23 行用红色的笔画出了外接圆, 其中 `abs(o - a)` 的结果是 O 点到 A 点的距离, 即外接圆的半径. 第 29 行在圆心 O 处画一个红色的点.

3.5 直角标记

在前面的程序中, 你已经看到 `Asymptote` 中定义了很多函数帮助我们完成一些特定工作. 这一节里我们准备亲自定义一个函数来画出中垂线和相应边之间的直角标记. 虽然在 `geometry`

模块中定义了 perpendicular 函数来画直角标记, 但为了了解函数的定义方法, 我们还是决定不用这个现成的, 而由自己来定义.

我们把这个函数起名为 rightangle, 并且希望它返回一个描述直角标记的路径, 这样就可以用 draw 函数来画它. 当我们描述一个角时, 我们需要 3 个点, 其中两个点位于这个角的两条边上, 而另一个则是它的顶点, 例如 ∠ODB. 因此我们的函数应该接受 3 个参数, 分别为这 3 个点. 另一方面, 我们还需要知道该把这个标记画多大, 因此还需要另外一个实型参数.

根据上面的思路, 我们可以把这个函数定义如下:

```
path rightangle(pair a, pair b, pair c, real size = 5bp) {
  pair ba = size * unit(a - b);
  pair bc = size * unit(c - b);
  pair bb = ba + bc;
  return shift(b) * (ba--bb--bc);
}
```

这个函数总共有 4 个参数, 其中最后一个 size 表示直角符号的大小. 注意, 我们为 size 指定了一个默认值, 就是说, 当调用这个函数时, 可以不给出最后这个参数 size 的大小, 在这种情况下, 它将取其默认值 5bp.

前 3 个参数是描述角的点, 我们用 a, b, 和 c 表示. 虽然这 3 个变量的名字和三角形的三个顶点的变量名相同, 但它们是不同的变量. 这里的 a, b, 和 c 是以函数参数的形式出现的, 它们仅仅在函数体内（两个花括号{}之间）才有意义, 因此是*局部变量*. 当函数被调用时, 例如 rightangle((1cm,1cm), (2cm,2cm), (1cm,3cm)) , 参数 a, b, 和 c 将分别被赋值为(1cm,1cm), (2cm,2cm), 和 (1cm,3cm). 但这并不会改变三角形三个顶点的坐标, 这正是由于函数的参数与函数之外定义的三角形顶点是不同的变量. 函数内局部变量的生存期仅限于函数体内部, 当函数调用结束并返回时, 局部变量所占用的内存将被系统收回.

> 三角形的顶点是在函数之外定义的, 变量名也是 a, b, 和 c. 它们是*全局变量*. 但由于它们的名字刚好跟我们函数中的局部变量重名, 因此在函数体内将不能被访问, 或者说, 它们被相应的同名局部变量屏蔽了. 见第 4.3 节.

在函数体中, 我们定义了 3 个点. 象参数变量一样, 它们也是局部变量. 其中 ba 是沿着 a - b 方向并且与坐标原点的距离为 size 的点. 这里调用 unit(a - b) 函数的结果为在 a - b 方向上的单位向量（一个点也可以理解为从坐标原点指向该点的向量）. 而 ba + bc 则可以理解为向量加法. 因此, 路径 ba--bb--bc 就是一个直角标记, 只不过它的位置在坐标原点. 于是, 我们在最后不直接返回它, 而是把这个路径平移到 b 点之后返回, shift 函数帮我们做了平移的工作.

现在, 我们程序的最新版本变为（输出结果如图 3.6 所示）:

```
1 import math;
2
3 // 辅助线
4 pen helpline = linewidth(0.2bp) + gray(0.5);
5
6 // 背景网格
```

```
for (int i = 0; i <= 8; ++i) {
  real x = i * cm;
  // 横线
  draw((0,x)--(8cm,x), helpline);
  // 竖线
  draw((x,0)--(x,8cm), helpline);
}

// 三角形 ABC
pair a = (2cm,1cm), b = (7cm,3cm), c = (3cm, 7cm);
draw(a--b--c--cycle);

// 中垂线、外心、外接圆
pair d = 0.5(a + b), e = 0.5(b + c), f = 0.5(c + a);
pair o = extension(d, rotate(90, d) * b, e, rotate(90, e) * b);

draw(circle(o, abs(o - a)), red);

draw(o--d, blue);
draw(o--e, blue);
draw(o--f, blue);

dot(o, red);

// 直角
path rightangle(pair a, pair b, pair c, real size = 5bp) {
  pair ba = size * unit(a - b);
  pair bc = size * unit(c - b);
  pair bb = ba + bc;
  return shift(b) * (ba--bb--bc);
}

draw(rightangle(b, d, o), blue);
draw(rightangle(b, e, o), blue);
draw(rightangle(c, f, o), blue);
```

3.6 文字标签

在图中添加文字标签非常简单, `label` 专门做这个. 你可以按照如下的方式简单地调用它:

```
label("字符串", 点);
label("字符串", 点, 方向);
```

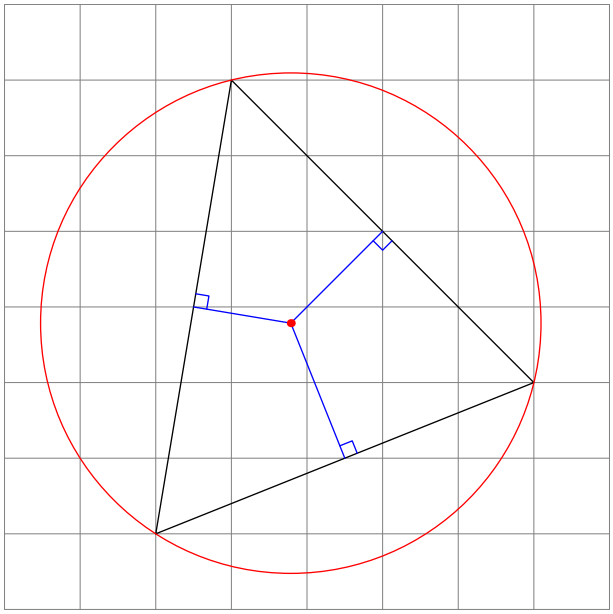


图 3.6: 添加直角标记.

这里, 双引号中的字符串就是要添加的文字内容, 点 表示添加文字的位置, 方向则是文字相对于点的方向. Asymptote 已经定义好了一些方向常数, 例如 E, S, W, N 分别表示东、南、西、北, 而 SE 则是东南, 等等.

对于较多的文字, 可以使用 minipage 函数, 它实际上就是 LaTeX 中的小页环境. minipage 函数接收两个参数, 第一个是文字内容, 第二个是小页的排版宽度.

需要注意的是, 所有的文字都会被送往 LaTeX 进行排版. 因此, 文字中可以包含 LaTeX 格式的数学公式. 当然也可以使用中文, 但要记得使用 CJK 宏包.

最终的程序列在下面, 它的输出就是图 3.1.

```
1  import math;
2  texpreamble("\usepackage{CJK}\AtBeginDocument{\begin{CJK*}{GBK}{song}}
3    \AtEndDocument{\clearpage\end{CJK*}}");
4
5  // 辅助线
6  pen helpline = linewidth(0.2bp) + gray(0.5);
7
8  // 背景网格
9  for (int i = 0; i <= 8; ++i) {
10   real x = i * cm;
11   // 横线
12   draw((0,x)--(8cm,x), helpline);
13   // 竖线
14   draw((x,0)--(x,8cm), helpline);
15 }
16
```

```
// 三角形 ABC
pair a = (2cm,1cm), b = (7cm,3cm), c = (3cm, 7cm);
draw(a--b--c--cycle);

// 中垂线、外心、外接圆
pair d = 0.5(a + b), e = 0.5(b + c), f = 0.5(c + a);
pair o = extension(d, rotate(90, d) * b, e, rotate(90, e) * b);

draw(circle(o, abs(o - a)), red);

draw(o--d, blue);
draw(o--e, blue);
draw(o--f, blue);

dot(o, red);

// 直角
path rightangle(pair a, pair b, pair c, real size = 5bp) {
  pair ba = size * unit(a - b);
  pair bc = size * unit(c - b);
  pair bb = ba + bc;
  return shift(b) * (ba--bb--bc);
}

draw(rightangle(b, d, o), blue);
draw(rightangle(b, e, o), blue);
draw(rightangle(c, f, o), blue);

// 标签
label("$A$", a, SW);
label("$B$", b, SE);
label("$C$", c, NW);
label("$D$", d, S);
label("$E$", e, NE);
label("$F$", f, NW);
label("$O$", o, SW);

// 文字
label(minipage("$\triangle ABC$ 的外接圆\ $\odot O$.\\
三角形的外心是三角形三条边的垂直平分线的交点.", 2.5cm), (8.2cm,5cm), E);
```