**1. GDI 是什么的英文缩写？**

**解答**:  
GDI 是 **Graphics Device Interface**（图形设备接口）的缩写。

* 它是 Windows 操作系统提供的一个图形系统，负责将应用程序的绘图指令转换为硬件设备（如显示器、打印机）可以执行的命令，实现了设备无关性的绘图功能。

**2. 什么是设备的无关性？**

**解答**:  
设备的无关性是指 GDI 允许程序员使用统一的绘图接口（GDI 函数），而不必关心底层硬件设备的具体实现或差异。

* 例如，你可以用相同的代码在显示器上绘制图形，也可以在打印机上输出相同的图形。
* GDI 负责将逻辑坐标和图形操作映射到不同的设备坐标系，屏蔽了设备分辨率、像素大小等硬件细节，简化了开发工作。

**3. GDI 的主要任务是什么？起到什么样的作用？**

**解答**:  
**主要任务**:  
GDI 的主要任务是提供一个通用的图形接口，处理图形绘制、文本显示和图像操作，并确保这些操作在不同硬件设备上保持一致。

**作用**:

* **绘制图形**: 支持绘制直线、矩形、圆形、曲线等基本图形。
* **填充区域**: 使用画刷填充封闭区域。
* **文本输出**: 显示和格式化文本。
* **图像处理**: 处理位图（如加载、显示、缩放）。
* **设备无关性**: 使程序在显示器、打印机等设备上都能正确渲染图形，无需针对每个设备编写特定代码。
* **效率优化**: 通过无效区域机制，只重绘需要更新的部分，提高性能。

**4. 什么是设备描述表？它的作业是什么？**

**解答**:  
**什么是设备描述表**:  
设备描述表（通常指设备上下文，英文 **Device Context**，简称 DC）是一个数据结构，包含了与特定设备（如窗口、打印机）相关的绘图状态和属性信息。

* DC 是 GDI 操作的核心，通过它可以访问设备的绘图功能，比如画笔、画刷、字体等。

**作业（作用）**:

* **存储绘图状态**: 保存当前的画笔、画刷、颜色、坐标系等设置。
* **提供绘图接口**: 程序通过 DC 调用 GDI 函数（如 LineTo、Rectangle）进行绘制。
* **设备通信**: 将绘图指令传递给硬件设备，确保正确渲染。
* 例如，获取一个窗口的 DC 后，可以用它画线或填充区域。

**5. 图形刷新包括什么？**

**解答**:  
图形刷新是指更新窗口或设备上显示内容的流程，通常包括以下内容：

* **清除旧内容**: 擦除无效区域的旧图形（可选，由 WM\_ERASEBKGND 消息处理）。
* **重绘图形**: 根据当前状态，使用 GDI 函数重新绘制窗口内容。
* **处理无效区域**: 只更新被标记为“无效”的部分，而不是整个窗口。
* **更新显示**: 将绘制结果呈现到屏幕或设备上。  
  图形刷新的核心是通过 WM\_PAINT 消息触发的重绘操作。

**6. 刷新请求有哪些？**

**解答**:  
刷新请求是指触发窗口重绘的条件或事件，通常包括：

* **窗口移动或调整大小**: 窗口部分被覆盖或暴露后需要重绘。
* **内容变化**: 应用程序修改了窗口内容（如绘制新图形）。
* **窗口最小化或恢复**: 恢复时需要重新绘制。
* **外部事件**: 其他窗口覆盖后移开，暴露隐藏区域。
* 这些事件会使 Windows 系统标记部分区域为“无效”，触发刷新。

**7. 应用程序情况下会发生刷新请求？同时发出什么消息？**

**解答**:  
**发生刷新请求的情况**:

* 应用程序主动调用 InvalidateRect 或 InvalidateRgn 标记窗口区域为无效。
* 窗口内容被修改（如画新图形、清除区域）。
* 窗口被其他窗口覆盖后重新暴露。

**同时发出什么消息**:

* 系统会发送 **WM\_PAINT** 消息到应用程序的消息队列。
* 程序在处理 WM\_PAINT 时调用 BeginPaint 获取 DC，进行重绘。

**8. 什么是无效区域？**

**解答**:  
无效区域（Invalid Region）是指窗口中需要重绘的部分区域。

* 当窗口发生移动、调整大小、被覆盖后暴露等事件时，Windows 会记录这些需要更新的区域。
* 无效区域由系统维护，WM\_PAINT 处理时，BeginPaint 会返回只包含无效区域的 DC，程序只需重绘这些部分，提高效率。

**9. 有效的刷新方法有哪些？大家觉得哪种比较好？在代码中若主动要求刷新窗口，应该怎么办？**

**解答**:  
**有效的刷新方法**:

1. **基于无效区域的刷新**:
   * 使用 InvalidateRect 或 InvalidateRgn 标记需要重绘的区域，触发 WM\_PAINT 消息，BeginPaint 只重绘无效区域。
2. **直接重绘**:
   * 使用 GetDC 获取整个窗口 DC，立即绘制内容，完成后用 ReleaseDC 释放。

**大家觉得哪种比较好**:

* **基于无效区域的刷新**更受欢迎：
  + 效率高，只重绘必要区域。
  + 减少闪烁，符合 Windows 机制。
* 直接重绘适合实时绘制（如动画），但可能导致闪烁，需配合双缓冲。

**主动要求刷新窗口怎么办**:

* 调用 InvalidateRect(hwnd, &rect, TRUE)（TRUE 表示擦除背景），或 InvalidateRect(hwnd, NULL, TRUE)（重绘整个窗口）。
* 示例：

c

InvalidateRect(hWnd, NULL, TRUE); // 标记整个窗口为无效

**10. 获取设备环境常用的两种方法是什么？**

**解答**:  
常用的获取设备上下文 (DC) 方法：

1. **BeginPaint**:
   * 在 WM\_PAINT 消息处理中使用，与 EndPaint 配对。
   * 返回只包含无效区域的 DC，适合重绘。
2. **GetDC**:
   * 可以在任何时候使用，返回整个窗口的 DC，需与 ReleaseDC 配对。
   * 适合非 WM\_PAINT 场景的即时绘制。

**11. 调用 BeginPaint 函数和 GetDC 有什么联系与区别？在实际编程中怎样选择使用？**

**解答**:  
**联系**:

* 两者都用于获取设备上下文 (DC)，是 GDI 绘图的入口。

**区别**:

* **使用场景**:
  + BeginPaint：仅限 WM\_PAINT 消息处理，自动处理无效区域。
  + GetDC：任何时候可用，返回整个窗口 DC。
* **无效区域**:
  + BeginPaint：只返回无效区域的 DC，效率高。
  + GetDC：返回整个 DC，无优化。
* **资源管理**:
  + BeginPaint 需配 EndPaint。
  + GetDC 需配 ReleaseDC。

**怎样选择使用**:

* **用 BeginPaint**:
  + 在 WM\_PAINT 中重绘时使用，推荐方式，避免闪烁。
* **用 GetDC**:
  + 非 WM\_PAINT 场景（如鼠标事件实时绘制）使用，但需注意释放 DC，并可能用双缓冲避免闪烁。
* **建议**: 优先用 BeginPaint，实时绘制时用 GetDC 并优化。

**12. 什么是映像模式？想一想，为什么要用映像模式？**

**解答**:  
**什么是映像模式**:  
映像模式（Mapping Mode）是 GDI 定义的逻辑坐标与设备坐标映射规则，决定了程序使用的单位如何转换为设备像素。

**为什么用映像模式**:

* **灵活性**: 允许用自然单位（如毫米、英寸）绘图，而非固定像素。
* **缩放和变换**: 方便实现图形的缩放、平移、旋转。
* **设备无关性**: 适应不同设备分辨率，程序无需调整代码。
* 例如，打印时用毫米单位，屏幕上自动按像素转换。

**13. 了解几种常用的映像模式，默认是哪种映像模式？**

**解答**:  
**常用的映像模式**:

* MM\_TEXT: 1 逻辑单位 = 1 像素，原点左上，X 右增，Y 下增。
* MM\_LOMETRIC: 1 单位 = 0.1 毫米。
* MM\_HIMETRIC: 1 单位 = 0.01 毫米。
* MM\_LOENGLISH: 1 单位 = 0.01 英寸。
* MM\_HIENGLISH: 1 单位 = 0.001 英寸。
* MM\_TWIPS: 1 单位 = 1/1440 英寸。
* MM\_ISOTROPIC: 保持 X、Y 比例相同，可自定义单位。
* MM\_ANISOTROPIC: X、Y 比例可不同，可自定义单位。

**默认映像模式**:

* 默认是 **MM\_TEXT**，最简单，直接对应像素。

**14. 了解映像模式中函数的使用（不需要书面写），可以写部分重点函数的理解。**

**解答**:

* **SetMapMode(hdc, mode)**: 设置当前映射模式（如 MM\_LOMETRIC），定义坐标单位。
* **SetWindowExt(hdc, x, y)**: 设置逻辑窗口的宽度和高度，影响缩放。
* **SetViewportExt(hdc, x, y)**: 设置设备视窗的宽度和高度，调整映射比例。
* **SetWindowOrg(hdc, x, y)**: 设置逻辑坐标原点。
* **SetViewportOrg(hdc, x, y)**: 设置设备坐标原点。
* **理解**: 这些函数通过调整逻辑和设备坐标的关系，实现图形的缩放、平移，适合需要精确控制的场景。

**15. 画笔的操作有哪些？**

**解答**:  
画笔（Pen）操作包括：

* **创建画笔**: 用 CreatePen 或 CreatePenIndirect 创建画笔，设置宽度、样式、颜色。
* **选择画笔**: 用 SelectObject(hdc, hPen) 将画笔选入 DC。
* **绘制**: 用 MoveToEx 和 LineTo 绘制线条。
* **删除画笔**: 用 DeleteObject(hPen) 释放资源。
* **修改属性**: 通过 SetDCPenColor 改变画笔颜色。

**16. 了解画笔中使用的函数名字的英文意思，便于大家记忆函数，（不需要书面写出）**

**解答**:

* **CreatePen**: "创建画笔"，用于生成新画笔对象。
* **CreatePenIndirect**: "间接创建画笔"，通过结构传入参数。
* **SelectObject**: "选择对象"，将画笔选入 DC 使用。
* **MoveToEx**: "移动到"，移动画笔起点。
* **LineTo**: "线到"，从当前位置画线到目标点。
* **DeleteObject**: "删除对象"，释放画笔内存。
* **SetDCPenColor**: "设置 DC 画笔颜色"，动态改变颜色。
* **记忆技巧**: 看函数名中的动词（如 Create、Select、Move）结合对象（Pen、Line），理解其作用。

**17. 了解常用的绘图函数，尤其是饼图函数，椭圆弧线函数。想一想怎么画圆，有没有画圆的函数？**

**解答**:  
**常用的绘图函数**:

* **饼图函数**:
  + Pie(hdc, left, top, right, bottom, x1, y1, x2, y2): 绘制扇形（饼图的一部分），参数定义外接矩形和起始/结束角度。
* **椭圆弧线函数**:
  + Arc(hdc, left, top, right, bottom, x1, y1, x2, y2): 绘制椭圆弧，参数定义外接矩形和起始/结束点。
  + Chord(hdc, left, top, right, bottom, x1, y1, x2, y2): 绘制椭圆弦（弧线两端连线）。

**怎么画圆，有没有画圆的函数**:

* **有画圆的函数**:
  + Ellipse(hdc, left, top, right, bottom) 可以画圆，当 left - right 和 top - bottom 相等时，绘制的是圆（特殊椭圆）。
* **画圆方法**:
  1. 定义一个正方形区域（如 left=50, top=50, right=100, bottom=100）。
  2. 调用 Ellipse(hdc, 50, 50, 100, 100)，即可画一个以 (75, 75) 为中心、半径 25 的圆。
* **注意**: 确保外接矩形是正方形，否则画出的是椭圆。