**自动化系统部署工具项目文档**

**1. 项目概述**

本工具用于在Windows PE环境中根据硬件信息自动选择并安装对应的操作系统镜像，支持自定义规则配置和日志记录。

**2. 功能特性**

* **硬件信息检测**：自动获取CPU型号、主板信息、内存容量。
* **规则匹配**：通过JSON配置文件动态匹配系统镜像。
* **自动化部署**：启动后自动执行部署逻辑，无需用户交互。
* **日志记录**：记录部署过程中的关键操作和错误信息。
* **自动重启**：部署完成后自动重启计算机。

#### ****3. 项目结构****

MyDeployTool/

├── SystemImages/ # 系统镜像目录

│ ├── Intel\_Win11Pro\_24H2.wim

│ ├── AMD\_Win11Pro\_24H2.wim

│ └── ...

├── rules.json # 规则配置文件

├── deploy.log # 日志文件

├── Hardware/

│ └── HardwareScanner.cs # 硬件信息检测模块

├── Deployment/

│ └── BackupManager.cs # 系统部署和备份模块

├── Rules/

│ └── RuleEngine.cs # 规则解析引擎

└── MainForm.cs # 主界面逻辑

#### ****4. 代码实现****

##### ****4.1 硬件信息模块 (****HardwareScanner.cs****)****

using System;

using System.Management;

namespace MyDeployTool.Hardware

{

public class HardwareScanner : IDisposable

{

private ManagementObjectSearcher \_cpuSearcher;

private ManagementObjectSearcher \_motherboardSearcher;

private ManagementObjectSearcher \_memorySearcher;

public string GetCPUInfo()

{

try

{

\_cpuSearcher = new ManagementObjectSearcher("SELECT Name FROM Win32\_Processor");

foreach (ManagementObject obj in \_cpuSearcher.Get())

{

return obj["Name"]?.ToString() ?? "Unknown";

}

}

catch (Exception ex)

{

LogError("获取CPU信息失败", ex);

}

return "Unknown";

}

public int GetTotalMemoryGB()

{

try

{

\_memorySearcher = new ManagementObjectSearcher("SELECT Capacity FROM Win32\_PhysicalMemory");

ulong total = 0;

foreach (ManagementObject obj in \_memorySearcher.Get())

{

total += Convert.ToUInt64(obj["Capacity"]);

}

return (int)(total / 1073741824); // 转换为GB

}

catch (Exception ex)

{

LogError("获取内存信息失败", ex);

return 0;

}

}

public void Dispose()

{

\_cpuSearcher?.Dispose();

\_motherboardSearcher?.Dispose();

\_memorySearcher?.Dispose();

}

private void LogError(string message, Exception ex)

{

File.AppendAllText("deploy.log", $"[{DateTime.Now}] {message}: {ex.Message}\n");

}

}

}

##### ****4.2 规则引擎 (****RuleEngine.cs****)****

using System;

using System.IO;

using Newtonsoft.Json;

using System.Linq.Dynamic.Core;

namespace MyDeployTool.Rules

{

public class RuleEngine

{

private class RuleConfig { public Rule[] Rules { get; set; } }

private class Rule { public string Condition { get; set; } public string ImagePath { get; set; } }

private readonly Rule[] \_rules;

public RuleEngine()

{

string configPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "rules.json");

if (!File.Exists(configPath)) throw new FileNotFoundException("规则文件未找到");

string json = File.ReadAllText(configPath);

\_rules = JsonConvert.DeserializeObject<RuleConfig>(json).Rules;

}

public string GetImagePath(string cpu, int memoryGB)

{

foreach (var rule in \_rules)

{

var parameters = new { CPU = cpu, MemoryGB = memoryGB };

var expr = DynamicExpressionParser.ParseLambda(

parameters.GetType(), typeof(bool), rule.Condition);

bool isMatch = (bool)expr.Compile().DynamicInvoke(parameters);

if (isMatch) return rule.ImagePath;

}

throw new InvalidOperationException("无匹配规则");

}

}

}

##### ****4.3 主程序逻辑 (****MainForm.cs****)****

using System;

using System.Windows.Forms;

using System.IO;

using MyDeployTool.Hardware;

using MyDeployTool.Rules;

namespace MyDeployTool

{

public partial class MainForm : Form

{

private readonly HardwareScanner \_hardware = new HardwareScanner();

private readonly RuleEngine \_ruleEngine = new RuleEngine();

private readonly Timer \_timer = new Timer { Interval = 5000 };

public MainForm()

{

InitializeComponent();

\_timer.Tick += (s, e) => StartDeployment();

\_timer.Start();

LoadHardwareInfo();

}

private void LoadHardwareInfo()

{

lblCPU.Text = $"CPU: {\_hardware.GetCPUInfo()}";

lblMemory.Text = $"内存: {\_hardware.GetTotalMemoryGB()} GB";

}

private void StartDeployment()

{

\_timer.Stop();

try

{

string imagePath = \_ruleEngine.GetImagePath(\_hardware.GetCPUInfo(), \_hardware.GetTotalMemoryGB());

if (File.Exists(imagePath))

{

new BackupManager().DeploySystem(imagePath);

MessageBox.Show("部署成功，即将重启...");

RebootComputer();

}

else MessageBox.Show($"镜像缺失: {imagePath}");

}

catch (Exception ex)

{

MessageBox.Show($"错误: {ex.Message}");

}

Application.Exit();

}

private void RebootComputer()

{

System.Diagnostics.Process.Start("shutdown", "/r /t 0");

}

}

}

#### ****5. 配置文件示例 (****rules.json****)****

{

"Rules": [

{

"Condition": "CPU.Contains(\"Intel Core\") && MemoryGB >= 16",

"ImagePath": "SystemImages/Intel\_Win11Pro\_24H2.wim"

},

{

"Condition": "CPU.Contains(\"AMD Ryzen\") && MemoryGB >= 16",

"ImagePath": "SystemImages/AMD\_Win11Pro\_24H2.wim"

},

{

"Condition": "MemoryGB < 16",

"ImagePath": "SystemImages/Default\_Win10Pro.wim"

}

]

}

**6. 部署说明**

1. **环境准备**：
   * 将程序与SystemImages、rules.json打包到Windows PE镜像。
   * 确保PE环境中已集成.NET Framework 4.8或更高版本。
2. **运行流程**：
   * 启动PE环境后自动运行程序。
   * 程序检测硬件信息 → 匹配规则 → 部署系统 → 重启。
3. **日志查看**：
   * 日志文件deploy.log记录详细操作过程，路径为程序运行目录。

**7. 常见问题处理**

* **规则不匹配**：检查rules.json中的条件语法和硬件信息是否准确。
* **镜像缺失**：确保SystemImages目录中存在对应的WIM文件。
* **权限不足**：在PE环境中以SYSTEM权限运行程序。

**项目源码下载**： https://github.com/HB30hz/MyDeployTool  
**联系支持**： ubiead@163.com