Bash 脚本中的错误处理与信号捕获机制

杨其臻

May 26, 2025

在自动化运维领域，Bash 脚本作为基础设施管理的重要工具，其稳定性和健壮性直接影响系统可靠性。当脚本因未处理的错误意外终止时，可能导致资源泄漏、数据不一致甚至服务中断。通过系统的错误处理与信号捕获机制，开发者可以实现「优雅降级」—— 在异常发生时执行资源清理、记录错误日志并控制退出流程。本文将从退出状态码的基础概念出发，逐步深入探讨信号捕获与错误处理的进阶技巧。

1 Bash 错误处理基础

每个 Bash 命令执行后都会返回一个介于 0-255 的退出状态码（Exit Status），通过 `$?` 变量可获取该值。约定俗成中，0 表示成功，非零值代表不同类型的错误。例如 `grep` 命令在未找到匹配时返回 1，而权限不足时返回 2。通过 `exit 3` 可主动终止脚本并返回自定义状态码。
基础错误检查常采用短路运算符简化流程控制：

```
mkdir /data || { echo "目录创建失败"; exit 1; }
```

此处的 `||` 运算符会在 `mkdir` 失败时执行右侧的代码块。更精细的错误处理可通过 `if` 语句实现：

```
if ! tar -czf backup.tar.gz /var/log; then
    echo "压缩失败，错误码：$?"
    exit 1
fi
```

对于复杂脚本，建议启用严格模式以增强错误检测：

```
set -euo pipefail
```

其中 `-e` 使脚本在任意命令失败时立即退出，`-u` 防止使用未定义变量，`-o pipefail` 确保管道命令中任一环节失败即视为整个管道失败。

2 信号基础与捕获机制

信号是操作系统与进程通信的基本机制。当用户按下 Ctrl+C 时，系统会向脚本发送 SIGINT（信号编号 2）；`kill` 命令默认发送 SIGTERM（15），而 SIGKILL（9）则会强制终止进程。信号捕获的核心工具是 `trap` 命令：

```
trap 'cleanup_temp_files' EXIT
```

此例在脚本退出时（无论是正常结束还是被信号终止）都会调用 cleanup_temp_files 函数。捕获 SIGINT 可实现交互式终止：

```
trap 'handle_interrupt' SIGINT

handle_interrupt() {
    read -p "确认退出? (y/n)" -n 1 choice
    [[ "$choice" == "y" ]] && exit 1
    echo "继续执行␣..."
}
```

这里的 trap 将 SIGINT 信号绑定到自定义处理函数，用户需二次确认才会真正退出。

3 高级错误处理模式

结合 ERR 伪信号可捕获非零退出状态的事件：

```
trap 'log_error␣$LINENO' ERR

log_error() {
    echo "[ERROR]␣在行号␣$1␣发生错误：$BASH_COMMAND"
    exit 1
}
```

该机制会记录错误发生的行号与具体命令，特别适合调试复杂脚本。对于需要原子性操作的场景，可设计回滚逻辑：

```
process_files() {
    local files_processed=()

    for file in *.csv; do
        process "$file" || { rollback "${files_processed[@]}"; return 1; }
        files_processed+=("$file")
    done
}
```

若任一文件处理失败，则回滚已处理文件。这种模式在数据库事务或批量操作中尤为重要。

4 信号竞态与作用域问题

在子 Shell 中设置的 trap 不会影响父进程环境。例如：

```
( set -e; trap 'echo␣Child␣Exit' EXIT; exit 1 )
echo "父进程继续执行"
```

子 Shell 的退出不会触发父进程的 EXIT 处理。信号处理函数应尽量简短以避免竞态条件，必要时使用锁机制：

```
trap '[[␣-f␣/tmp/lock␣]]␣||␣handle_signal' SIGTERM
```

此处的文件锁确保信号处理函数不会并发执行。

5 最佳实践与调试技巧

推荐在脚本开头启用严格模式并初始化变量：

```
#!/bin/bash
set -euo pipefail
declare -g tmp_dir="/tmp/$(basename␣$0).$$"
```

关键操作应添加超时控制：

```
if ! timeout 30s long_running_task; then
   echo "任务超时"
   exit 1
fi
```

调试时可启用执行追踪：

```
export PS4='+[${LINENO}:${FUNCNAME[0]}]␣'
set -x
```

PS4 变量定义了调试信息的格式前缀，结合 `set -x` 可输出详细的执行过程日志。

完善的错误处理机制可使 Bash 脚本达到工业级可靠性标准。通过合理运用退出状态码、严格模式与信号捕获，开发者能够构建出具备自我修复能力的自动化工具。对于关键生产环境脚本，建议配合 ShellCheck 进行静态分析，并参考《Advanced Bash-Scripting Guide》等权威资料持续优化代码质量。