# 在Cisco ESA和CES中配置和验证正则表达式

## 目录



## 简介

本文档介绍ESA和CES如何在过滤器中使用正则表达式、关键行为差异以及实施前测试的必要性。

## 背景信息

本文档介绍当在邮件过滤器和内容过滤器中使用时，思科邮件安全设备(ESA)和思科云邮件安全(CES)如何处理正则表达式。它特别侧重于了解正则表达式在这些组件中的行为方式，以及它们如何与邮件信头、正文内容和附件交互。

务必从一开始就说明DLP模块中使用的正则表达式引擎的行为不同。因此，本文档中描述的所有内容都仅适用于邮件过滤器和内容过滤器，而不适用于DLP策略。

在ESA中使用正则表达式时，管理员必须了解电子邮件内容的计算方式与邮件客户端中的直观显示方式不同。电子邮件包含信封信息、结构化信头、MIME部分和可能编码的内容。因此，如果不能完全理解消息结构和正则表达式行为，过滤器执行的比较可能会产生意想不到的结果。

因此，在执行之前，任何使用正则表达式的新过滤器始终可以在监控模式下启用。这允许验证实际流量，并防止意外阻止或性能影响。

# 词典和搜索词

创建邮件过滤器或内容过滤器时，在许多条件中输入的术语会解释为正则表达式。这是一个关键概念：即使管理员打算匹配文字文本，ESA也可以使用regex逻辑处理输入。

这并不统一适用于所有条件类型。例如，当在某些结构化条件中搜索特定IP地址时，该值不会解释为正则表达式。但是，当在Subject信头、邮件正文、特定信头字段或附件文件名中进行搜索时，该值通常被视为regex模式。

一个常见示例清楚地说明了这一点。假设目标是阻止具有以下主题的电子邮件：

```
Receipt number (123456)
```

由于括号是正则表达式中的特殊字符（用于分组），因此必须将其转义。

正确的表达式是：

```
Receipt number \(123456\)
```

如果圆括号未转义，正则表达式引擎将它们解释为分组运算符，而不是文字字符。根据模式，这可能导致意外匹配或不同于预期的行为。

因此，必须了解哪些字符在regex中具有特殊含义，并确保在需要文字匹配时正确转义它们。

## 特殊字符及其转义语法示例

第一列显示包含特殊字符的示例文本，第二列显示如何编写正确的正则表达式语法以匹配Cisco ESA中的文本文本(Python-style regex)。

| 要匹配的文字文本 | 更正正则表达式语法 |
| --- | --- |
| 收据编号(123456) | 收据编号\(123456\) |
| user@example.com | user@example \.com |
| www.test.abc | www\.test\.abc |
| file_name.txt | file_name\.txt |
| 价格是10.50 | 价格是10\.50 |
| C:\Users\Admin | C:\\Users\\Admin |
| [机密] | \[机密\] |
| {invoice} | \{invoice\} |
| +34 600 123 456 | \+34 600 123 456 |
| 问题？ | 问题？ |
| 100%保证 | 100%保证（%不需要转义） |
| 星号*符号 | 星号*符号 |
| A\|B | A\\|B |
| 脱字符^start | 脱字符号\^开始 |
| 100美元 | 美元\$100 |

## 限制正则表达式的使用

正则表达式必须谨慎使用，并且仅在必要时使用。虽然它们提供强大的匹配功能，但设计不当或过多的表达式会增加消息处理时间并产生意外的匹配。

需要注意的一个特殊构造是。* ，它表示“任何字符，零次或多次”。 如果放置在表达式的开头或结尾，可能会导致过多的回溯和不必要的处理开销。

Cisco文档表明，在开头或结尾使用.*的条目可能会导致系统在匹配特定MIME部分时在某些条件下锁定。因此，思科建议尽可能避免使用前导或尾随。*。

在许多场景中，管理员使用.*invoice.*等模式，以便在ESA中简单编写发票并产生相同的实际结果。由于扫描引擎已搜索相关内容区域，因此围绕.*的词经常是冗余且计算效率低下的。

---

⚠ 警告：一般建议是尽可能保持正则表达式简单和精确。

---

## 邮件过滤器、内容过滤器和词典

Cisco ESA提供多种机制来评估消息和应用操作。邮件过滤器在管道的开始时运行，并使用脚本样式语法。它们非常灵活，允许涉及信封数据、信头和附件属性的高级逻辑。但是，由于它们在处理链的早期执行，低效的邮件过滤器可能会对性能造成负面影响。

内容过滤器通过图形界面配置，并在消息被接受后运行。对于大多数内容检测使用案例，从性能的角度来看，内容过滤器更易于管理且更安全。

在邮件过滤器和内容过滤器中，正则表达式既可以直接引入条件，也可以通过使用字典间接引入。

词典允许管理员集中使用可重复使用的搜索词。每个条目都写在单独的行上，可以是纯文本或正则表达式。词典也支持非ASCII字符，因此适合多语言环境。

在某些情况下，某些复杂的正则表达式结构在词典中无法行为相同。发生这种情况时，正则表达式必须直接放置在过滤器条件中而不是词典内部。

Cisco ESA允许创建多达150个内容词典。默认情况下，除非使用dictionaryconfig命令通过CLI修改限制，否则可配置100个词典。

词典还可以执行术语加权。每个术语可以分配一个数字权重，当ESA扫描消息时，它会将该术语的出现次数乘以其权重。生成的得分与过滤器中定义的阈值进行比较。此评分模型允许更灵活且分级的策略实施。

此外，词典可以包括智能标识符，它是用于结构化数字模式（如社会保险号码或银行标识符）的算法检测器。

# 正则表达式引擎

Cisco ESA使用基于Python re模块样式的正则表达式。虽然这提供了与常见Python regex语法的兼容性，但并非所有Python环境中支持的高级功能都一定在ESA中受支持。

为了进行精确的字符串匹配，表达式必须在开头使用^并在结尾使用$定位。如果没有这些锚点，正则表达式引擎可以匹配子字符串而不是完整值。

例如，表达式：

```
sun.com
```

匹配字符串，例如：

```
thegodsunocommando
```

但是，表达式：

```
^sun\.com$
```

仅与精确字符串sun.com匹配。

当匹配空字符串时，不要使用“”，因为这样会有效地匹配所有字符串。相反，正确的表达式是：

```
^$
```

由于Cisco ESA使用Python风格的正则表达式，因此可通过几种方法执行不区分大小写的比较。

默认情况下，如前所述，正则表达式区分大小写。这意味着搜索：

```
foo
```

只匹配foo，但不匹配FOO、Foo或fOo。

如果要执行不区分大小写的匹配，可以使用正则表达式开头的内联标志(?i)。这会告知regex引擎忽略模式其余部分的大小写。

例如：

```
(?i)foo
```

此表达式匹配：

- foo
- FOO
- foo
- fO

如果要完全匹配整个字符串（忽略大小写），可以将不区分大小写的标志与锚点组合起来：

```
(?i)^foo$
```

这样可确保无论资本化如何，全部价值都完全为“foo”。

另一个（不太实际的）替代方法是使用字符类显式定义所有可能的组合，例如：

```
[Ff][Oo][Oo]
```

但是，这种方法很难维护，建议不要使用(?i)标志来代替。

在大多数ESA场景中，区分大小写匹配的首选和最干净方法是使用：

```
(?i)
```

位于正则表达式开头。

# 非ASCII字符和字边界

在使用双字节字符集的语言中，词边界或大小写的概念不能按预期运行。当编码或区域设置未知时，依赖于\w等构造的复杂表达式会产生不一致的结果。

在这种情况下，建议禁用词典配置中的词边界实施，或简化表达式，以避免对模糊字符类的依赖。

使用非ASCII词典时，CLI显示无法正确呈现字符，具体取决于终端编码。在这种情况下，建议将词典导出到文本文件，在外部进行编辑，然后重新导入该词典。

# 编写高效过滤器

在写入过滤器时，效率至关重要，特别是在高容量环境中。一个常见错误是为类似的匹配项编写OR条件的长链。

例如，逐个检查数十个连接扩展会强制regex引擎重复初始化。这会增加CPU使用率并降低可维护性。

使用单个正则表达式中的交替方式将它们分组可显着减少处理开销，而不是编写许多单独的比较。这减少了调用regex引擎的次数，并使过滤器更易于维护。

高效的过滤器设计不仅与可读性有关，还直接影响系统性能。

# PDF和正则表达式

根据PDF的生成方式，匹配PDF文件中的内容可能会产生意外的结果。某些PDF的内部表示中不包含逻辑空格或换行符。扫描引擎尝试根据字词定位来重建逻辑间隔。

如果使用多种字体或字体大小构建单词，则内部表示可以对文本进行分片。例如，“callout”一词在内部可以解释为“call out”或“c a l lout”。

在这种情况下，尝试匹配表达式“callout”可能会失败，因为内部表示法不包含完全连续的字符串。管理员在设计针对PDF附件的基于内容的策略时必须注意这一限制。

# 测试正则表达式

在将正则表达式部署到生产环境之前对其进行测试是一项关键操作要求。当根据实际邮件流量进行评估时，语法正确的正则表达式的行为可能截然不同。如果没有正确的测试，过滤器可能会生成误报、无法检测预期模式、引入性能开销或无意中断合法邮件流。

测试必须采用结构化的两阶段流程，以便在生产中启用过滤器之前将风险降至最低。

第1阶段 — 正则表达式设计和验证
第一阶段侧重于设计和验证正则表达式本身，然后再将其集成到Cisco ESA。

1.使用regex101或类似工具

在线平台(例如http://regex101.com（或等效工具）在设计阶段非常有用。使用这些工具时，必须选择Python风格以接近ESA的regex引擎。

这些平台允许管理员：

- 验证语法正确性
- 确认特殊字符已正确转义
- 测试匹配和非匹配案例
- 可视化分组和量化器行为
- 识别潜在的贪婪结构，例如。*

但是，这些工具模拟标准Python regex行为，可支持思科ESA中未完全实施的功能。因此，它们必须被视为初步的验证工具，而不是最终的兼容性测试。

2.使用AI模型(ChatGPT、Copilot、...)

基于AI的助理可以加快正则表达式的创建，特别是在复杂的匹配场景中。通过以自然语言描述所需行为，管理员可以获取初始正则表达式提议，然后可以对其进行优化。

AI工具特别有助于：

- 生成复杂的分组表达式
- 将业务要求转换为regex语法
- 将长期基于OR的条件简化为分组选项

然而，人工智能产生的表达式必须始终以批判的眼光审视。它们可能导致低效率、不支持的结构或过于复杂的逻辑。人工智能援助必须被视为起草援助，而不是最终验证。每个AI生成的表达式仍必须使用结构化验证方法进行测试。

第2阶段 — 思科ESA中的过滤器行为验证

一旦表达式本身经过验证，第二阶段将重点确认其在应用于实际消息处理时在Cisco ESA内的行为。

1.使用CES控制台中的跟踪功能

Cisco Email Security(CES)控制台中的Trace（跟踪）功能允许管理员模拟和分析特定邮件的处理方式。这是在执行前验证过滤器行为的最可靠方法之一。

Trace提供以下方面的可视性：

- 消息解析方式
- 评估哪些过滤器
- 是否触发条件
- 规则执行的顺序

由于ESA执行MIME解析、报头规范化和内容解码，设备内的行为可能与外部regex测试工具不同。有关详细说明，管理员必须查阅官方Cisco文档：

https://www.cisco.com/c/en/us/td/docs/security/ces/ces_16-0-3/user_guide/b_ESA_Admin_Guide_ces_16-0-3/b_ESA_Admin_Guide_12_1_chapter_0101001.html

使用Trace可确保过滤器在实际处理引擎中按预期运行。

2.使用日志记录操作创建过滤器

另一种安全推荐的方法是使用无中断操作（如日志记录）部署过滤器，而不是使用主动操作（如丢弃、退回或隔离邮件）。

通过将过滤器配置为在匹配时记录条目，管理员可以：

- 观察匹配频率
- 检测意外触发器
- 验证性能影响
- 分析实际流量行为

此方法可有效地将过滤器放置在生产流量内的受控监控阶段。一旦完成足够的验证并确认行为正确，即可将操作安全地更改为实施模式。

## 在内容过滤器和词典中引入表达式

正确设计和验证正则表达式后，下一步是了解如何在Cisco ESA中输入正则表达式。根据表达式是

直接在内容过滤器条件中配置还是在“词典”中配置，语法可能会略有不同。这种差异经常造成混淆。

## 在内容过滤器中引入表达式

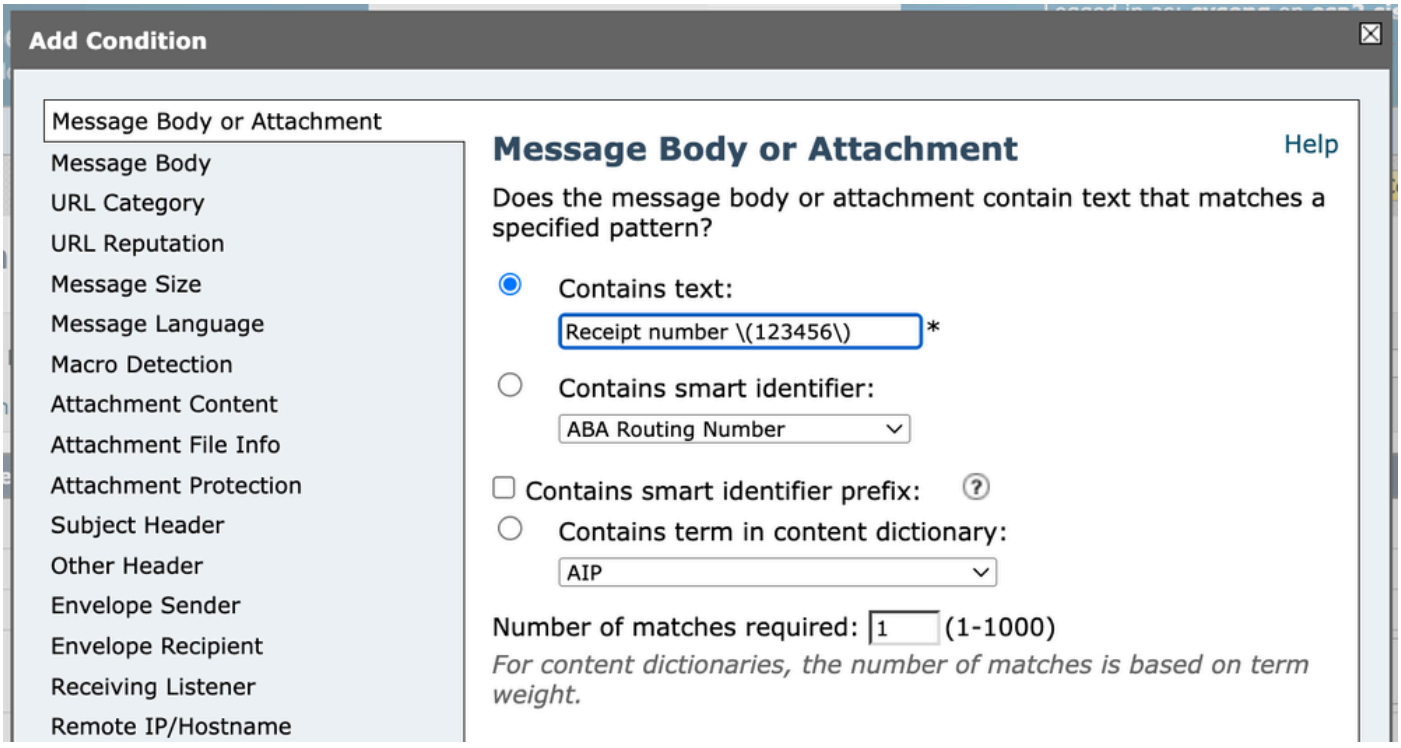
配置内容过滤器条件（例如，匹配主题信头）时，必须在条件字段中输入正则表达式。如果想要匹配文字文本：

```
Receipt number (123456)
```

必须转义括号，因为它们是正则表达式中的特殊字符。

因此，正则表达式本身必须写为：

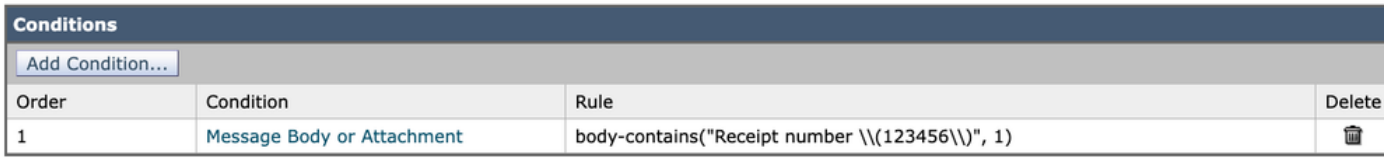
```
Receipt number \(123456\)
```




内容过滤器1

但是，在GUI或高级配置输出中查看完全过滤条件时，可能会显示为：

```
subject == "Receipt number \\(123456\\)"
```




内容过滤器2

乍一看，这可能令人困惑。双反斜杠(\\)的原因在于反斜杠本身也是带引号的字符串中的特殊字符。在此上下文中，一个反斜杠用于转义regex引擎的括号，而第二个反斜杠用于转义带引号的字符串中的反斜杠。

实际上：

\(123456\)是实际的正则表达式。

\\()是系统在带引号的配置字符串内表示\()的方式。

尽管显示时显示不同，但所计算的逻辑正则表达式仍为：

收据编号\(123456\)

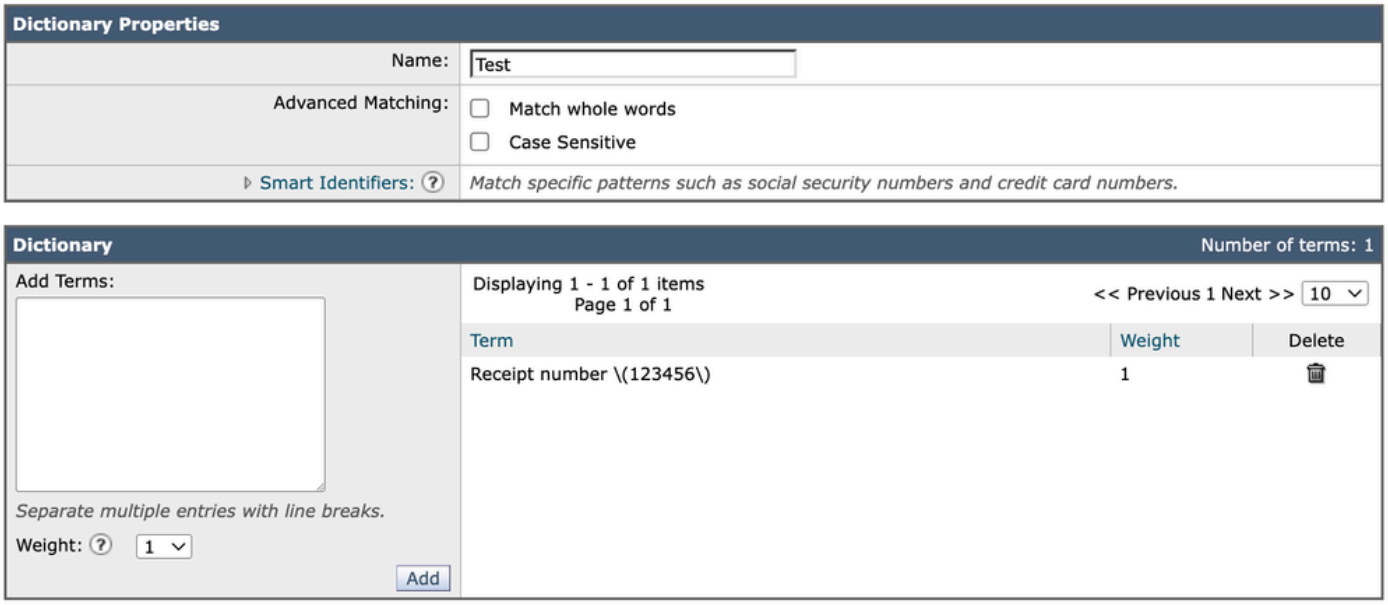
这只是配置输出中的字符串转义问题。

## 词典中的表达式简介

将同一表达式添加到词典时，该条目会按以下方式直接引入：

```
Receipt number \(123456\)
```

在这种情况下，系统仍会完全按照写入的内容显示。与内容过滤器GUI表示方式不同，词典不需要其可视配置格式的其他转义层。



字典

根据词典的结构，每条词典条目都会被评估为纯文本或正则表达式。如果包含特殊字符（如本例中的圆括号），输入表达式时必须已正确转义。

关于“匹配整字”

在配置词典时，有一个名为“匹配整个单词”的选项。 在许多情况下，建议在使用正则表达式时不要依赖此设置。

这是因为可以使用正则锚点更精确地控制字边界行为。

例如：

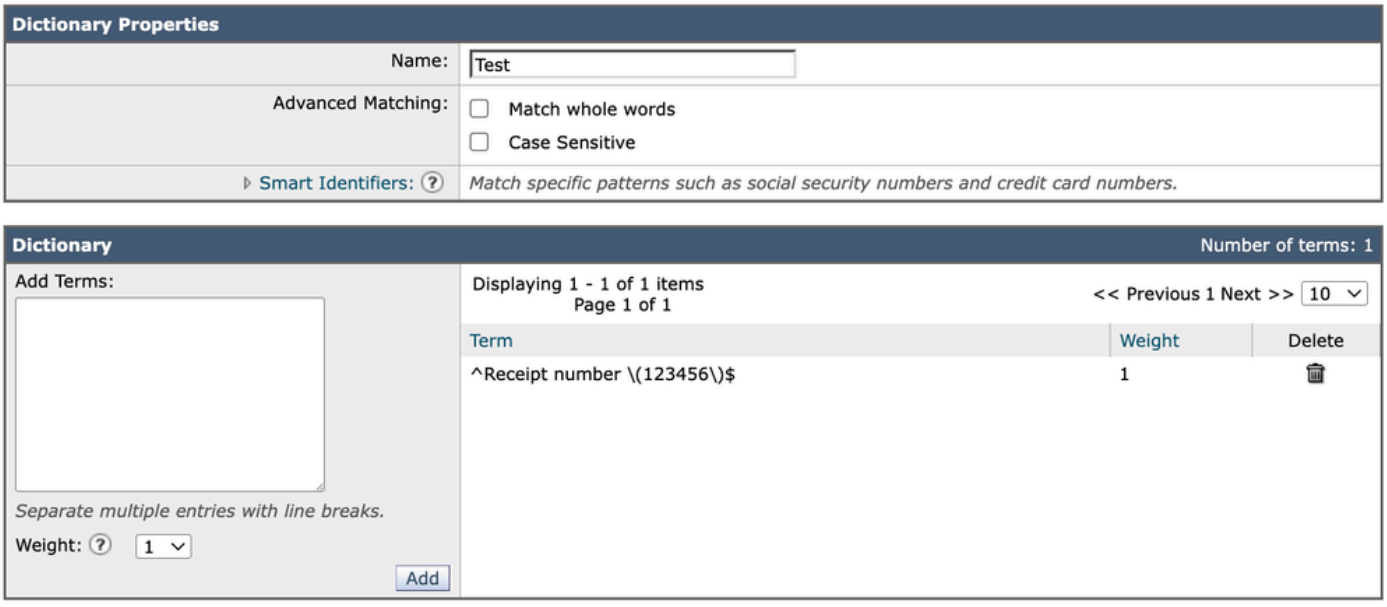
^确保匹配从开头开始。

$确保匹配在末尾结束。

使用锚点，例如：

```
^Receipt number \(123456\)$
```

对精确匹配行为提供明确且可预测的控制。这种方法避免了与字边界解释方式相关的潜在歧义，特别是在多语言或非ASCII环境中。




字典2

因此，通常最好直接在正则表达式中管理匹配精度，而不是依赖“匹配整字”选项。

了解内容过滤器和词典之间的这些细微差异可确保表达式的行为保持一致，并降低实施过程中出现配置错误的风险。

# 思科ESA中的Regex成本排名

在Cisco ESA中使用正则表达式时，性能影响主要取决于引擎必须扫描的文本数量以及必须执行回溯的文本数量。由于ESA必须评估整个邮件正文、MIME部分甚至已解码的附件，低效模式会显着增加CPU使用率。

它是从最高计算成本到最低的实际排序。

## 最昂贵 — 高风险模式

这些表达式可能会严重影响性能，尤其是对于大型邮件。

嵌套限定符（最坏情况）

示例：

```
(.*)+
(.+)+
(\S+)+
```

这些都是极其危险的，因为它们创造了指数级回溯情景。

另一个量词内的量词强制正则表达式引擎在失败之前尝试许多组合。

在实际流量中，这可能导致严重的CPU峰值。

建议：避免使用无限定和模糊的嵌套限定符。

贪婪。*后跟所需的模式

示例：

```
.*text
.*\/\?text
```

此模式首先使用整个消息，然后逐个字符回溯，直到找到所需的子字符串。

如果模式不存在（或接近结束），引擎会回溯并在多个位置测试所需的令牌，这会增加CPU成本。

在ESA中，正文可能很大，并且包含MIME内容，这很快变得非常昂贵。

建议：不要将。*预置为检测子字符串。ESA已搜索评估的内容，领先的通配符只会增加回溯和CPU使用率。

```
text$
\/\?text$
```

具有共享前缀的大型替代项

示例：

```
(a.*b|a.*c|a.*d)
```

当多个备选方案共享结构时，引擎将按顺序评估每个分支。

如果早期的分支几乎匹配，但出现延迟故障，引擎会大量重试。

这显着增加了评估时间。

中等成本 — 请谨慎使用

这些模式不是灾难性的，但仍可能效率低下。

广泛.*使用

示例：

```
https://.*\?text
```

虽然不是指数级匹配，.*仍允许无限匹配。如果预期的子字符串没有快速出现，引擎将扫描大部分消息。

在ESA中，当扫描邮件正文以查找网络钓鱼URL时，这种情况很常见。

延迟量词(+?,*?))

示例：

```
\S+?
.*?
```

延迟量词更改匹配策略（最短优先）。 它们可以在某些模式中减少过度匹配，但在大型“搜索”工作负载中，当终端令牌延迟或缺失时，它们可以增加尝试次数。

在许多ESA使用案例中，它们无法提供真正的优势，并且可能会引入不必要的内部重试。

非常通用的字符类

示例：

```
\S+
.+
```

这些选项允许较大的匹配范围，从而增加了潜在回溯路径的数量。

更具体的字符类始终更可取。

## 低成本 — 安全高效的模式

建议用于生产ESA环境。

固定文字

示例：

```
text
iw\.adc
```

文字字符串是最有效的匹配项。该引擎以最小的开销执行直接比较。

使用锚点限制搜索范围

当匹配预期位于特定位置时，请考虑使用^或$锚定模式。锚点将评估限制在固定位置，防止引擎不必要地扫描整个内容。这可减少回溯并提高性能，特别是在大型邮件正文或结构化信头中。

```
^Invoice$
```

特定字符类

```
[A-Za-z0-9.-]+
[^/\s]+
```

这些限制可以匹配的内容，从而显着减少搜索空间并限制回溯。

结构化和约束模式

示例：

```
https?:\/\/[A-Za-z0-9.-]+(?:\/[^?\s]*)*\/\?text
```

- 域已修复。
- 不使用。*。
- 不包含灾难性嵌套模式(例如，(.*)+)
- 没有不必要的懒惰操作员。
- 每个截面都受约束。

与广泛的通配符匹配相比，这显着降低了CPU影响。

## Cisco ESA实用指南

为邮件或内容过滤器设计regex时：

1. 模式越具体，性能就越好。
2. 请避免使用。*，除非确实必要，特别是避免在之后放置所需的令牌。
3. 切勿使用嵌套的限定符。
4. 首选显式字符类而不是通配符。
5. 在实施之前，始终在监控模式下测试新表达式。

## Regex性能比较（Cisco ESA环境）

| 模式 | 推荐 | 回溯风险 | ESA影响 | 推荐的替代方案 |
|---|---|---|---|---|
| https?:\/\/.*/\？文本。* | 无 | 高 | 更高 | ^https?:\/\/[A-Za-z0-9.-]+(?:\/[^?\s]*)*\/\？文本 |
| https?:\/\/.*\？文本 | ⚠小心 | 中高 | 中高 | ^https?:\/\/[^?\s]+\?text$ |
| https?:\/\/.* | 无 | 中高 | 中 | ^https?:\/\/[A-Za-z0-9.-]+(?:\/[^\s]*)* |
| .*密码 | 无 | 高 | 更高 | 密码$ |

| | | | | |
|---|---|---|---|---|
| .*文本。* | 无 | 高 | 更高 | 文本 |
| .*(invoice\|payment\|transfer) | 无 | 高 | 更高 | （发票\|付款\|转帐）$ |
| (.+)+ | 从不 | 极高（指数） | Severe（严重） | 不使用嵌套的限定符进行重构（例如。+） |
| .*@.* | 无 | 高 | 更高 | [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,} |
| \S+? | 不理想 | 中 | 中 | \S+或更具体的类，如[A-Za-z0-9.-]+ |
| .*\/admin | 无 | 高 | 更高 | \/admin$ |
| .*(login\|verify)。* | 无 | 高 | 更高 | （登录\|验证） |
| ^.*文本 | 无 | 高 | 更高 | text$(或^text（如果职位重要） |

# 结论

正则表达式是Cisco ESA中功能强大且灵活的工具，可在邮件过滤器和内容过滤器中实现精确的内容检测和高级策略实施。然而，这种灵活性意味着责任。设计欠佳或测试不足的表达式可能导致误报、漏检、性能下降或合法电子邮件流量的意外中断。

因此，在ESA中使用正则表达式必须始终采用结构化和规范化的方法。创建阶段必须确保表达式的语法正确、正确转义、有效且与预期目标在逻辑上保持一致。外部工具和AI辅助的生成可以大大加快这一过程，但它们绝不能取代仔细验证。

同样重要的是ESA环境本身的验证阶段。由于ESA通过MIME解析、报头规范化和内容解码来处理邮件，因此实际行为可能与理论预期不同。管理员最初在日志记录或监控模式下使用跟踪和部署过滤器等工具可以确认正确的行为，而不会产生操作风险。

总之，正则表达式必须保持尽可能简单，必须经过彻底的测试，而且必须谨慎部署。设计良好且经过适当验证的过滤器不仅可以有效地实施策略，还可以保护系统稳定性并确保生产环境中的可预见行为。

# 文档

有关正则表达式在Cisco ESA中如何实施和使用的其他技术细节和官方指南，管理员必须查阅Cisco产品文档

“规则中的正则表达式”部分概述了如何在邮件过滤器和内容过滤器中评估正则表达式，包括规则条件中的语法注意事项和用法。

https://www.cisco.com/c/en/us/td/docs/security/ces/ces_16-0-3/user_guide/b_ESA_Admin_Guide_ces_16-0-3/b_ESA_Admin_Guide_12_1_chapter_01000.html#con_1192755

“正则表达式使用指南”一节提供了一些实用建议，包括正确语法、锚定表达式、处理特殊字符，以及避免可能影响性能或匹配准确度的常见错误。

https://www.cisco.com/c/en/us/td/docs/security/ces/ces_16-0-3/user_guide/b_ESA_Admin_Guide_ces_16-0-3/b_ESA_Admin_Guide_12_1_chapter_01000.html#con_1125696

在设计和故障排除依赖于正则表达式的过滤器时，强烈建议查看这些官方资源，因为它们提供符合所使用的特定AsyncOS版本的权威性指导。

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。