数据传输服务

常见问题

产品文档

腾讯云

文档目录

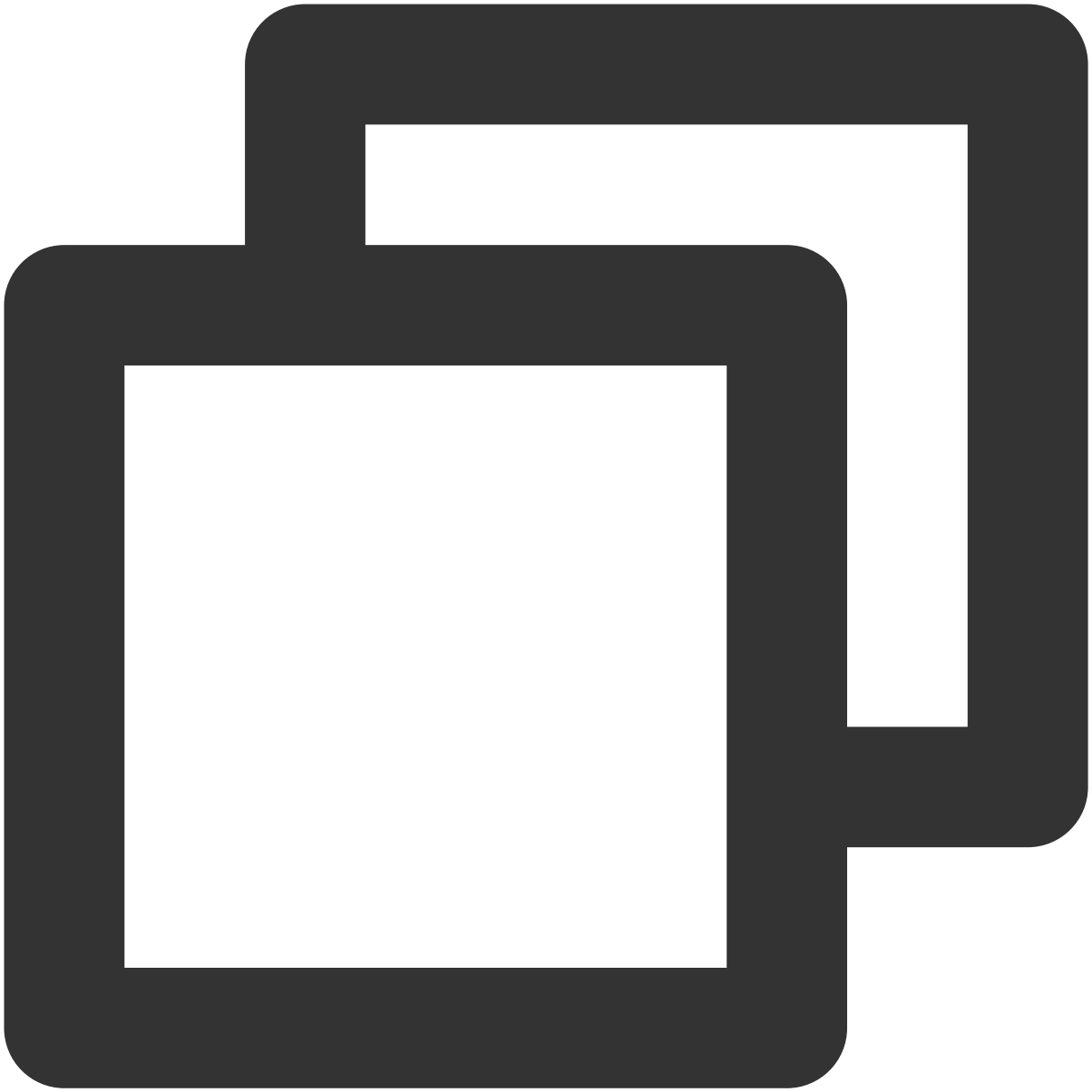
常见问题
数据迁移

最近更新时间：2024-07-08 15:36:57

通用问题

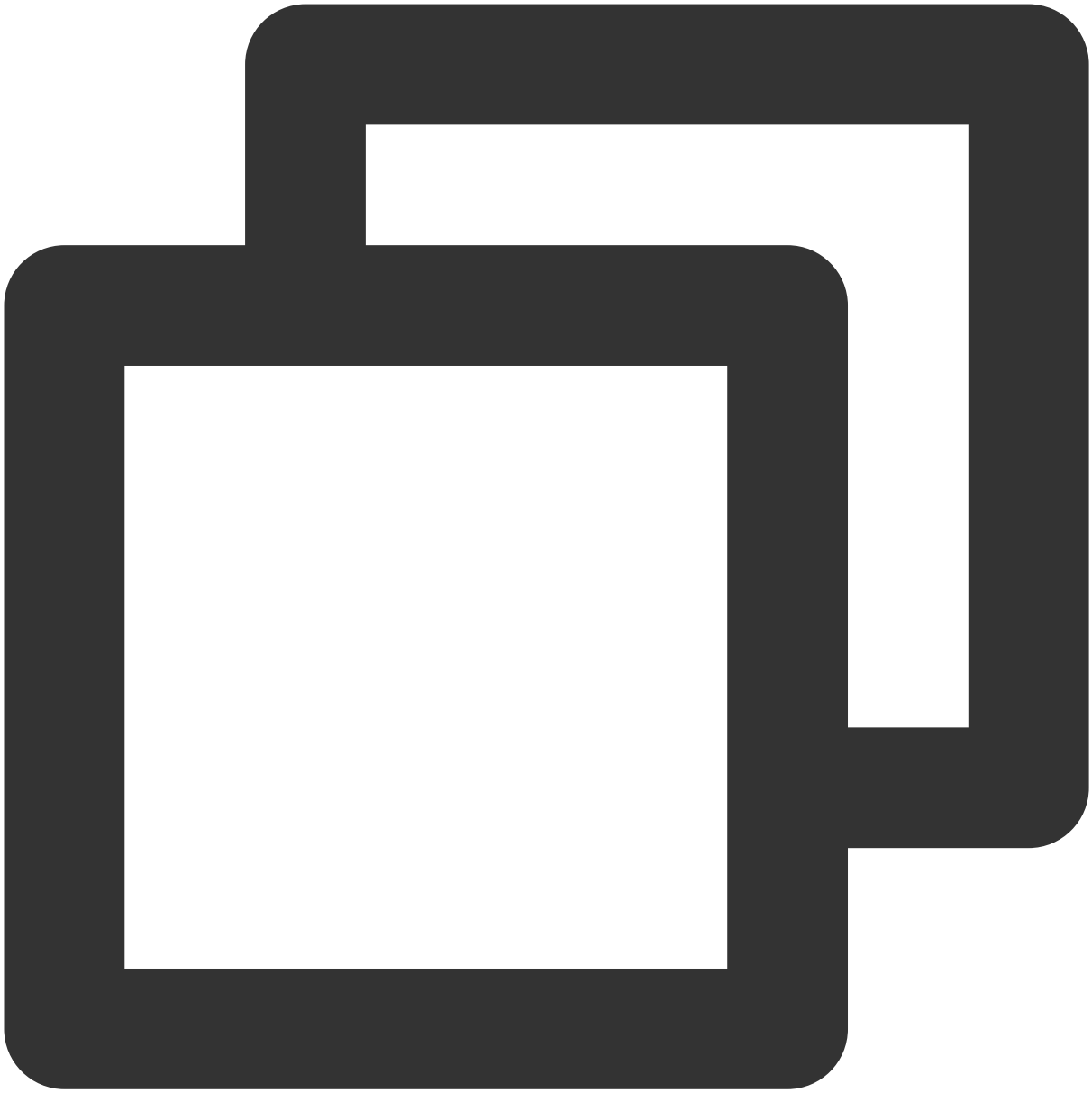
使用 DTS 进行数据迁移/同步，对源数据库有啥影响？

对源数据库的数据内容无影响，源库可正常进行业务写入。DTS 进行数据迁移时，其实是复制了源数据库的一份数据，不会删除源数据库的内容，所以对源数据库的数据没有影响。
对源数据库的性能有一定影响，**影响主要在 CPU**。DTS 在执行全量数据迁移/同步时，会将源库的全量数据全部读取一次，所以会增加源库自身的压力。
源库为 MySQL，规格8核16G，DTS 任务默认采用8线程并发，在网络无瓶颈的情况下，DTS 任务对源库的性能影响如下：
DTS 全量导出阶段：占用源库约18%-45%的 CPU，增加源库约40-60MB/s的查询压力，占用约8个活跃 session 连接数。
DTS 增量导出阶段：对源数据库基本无压力，只有一个连接实时监听源库的 binlog 日志。
DTS 全量阶段源库的各连接详情如下：
1. 任务开始阶段会有1个线程查询系统表获取导出信息，会有类似如下 SQL。



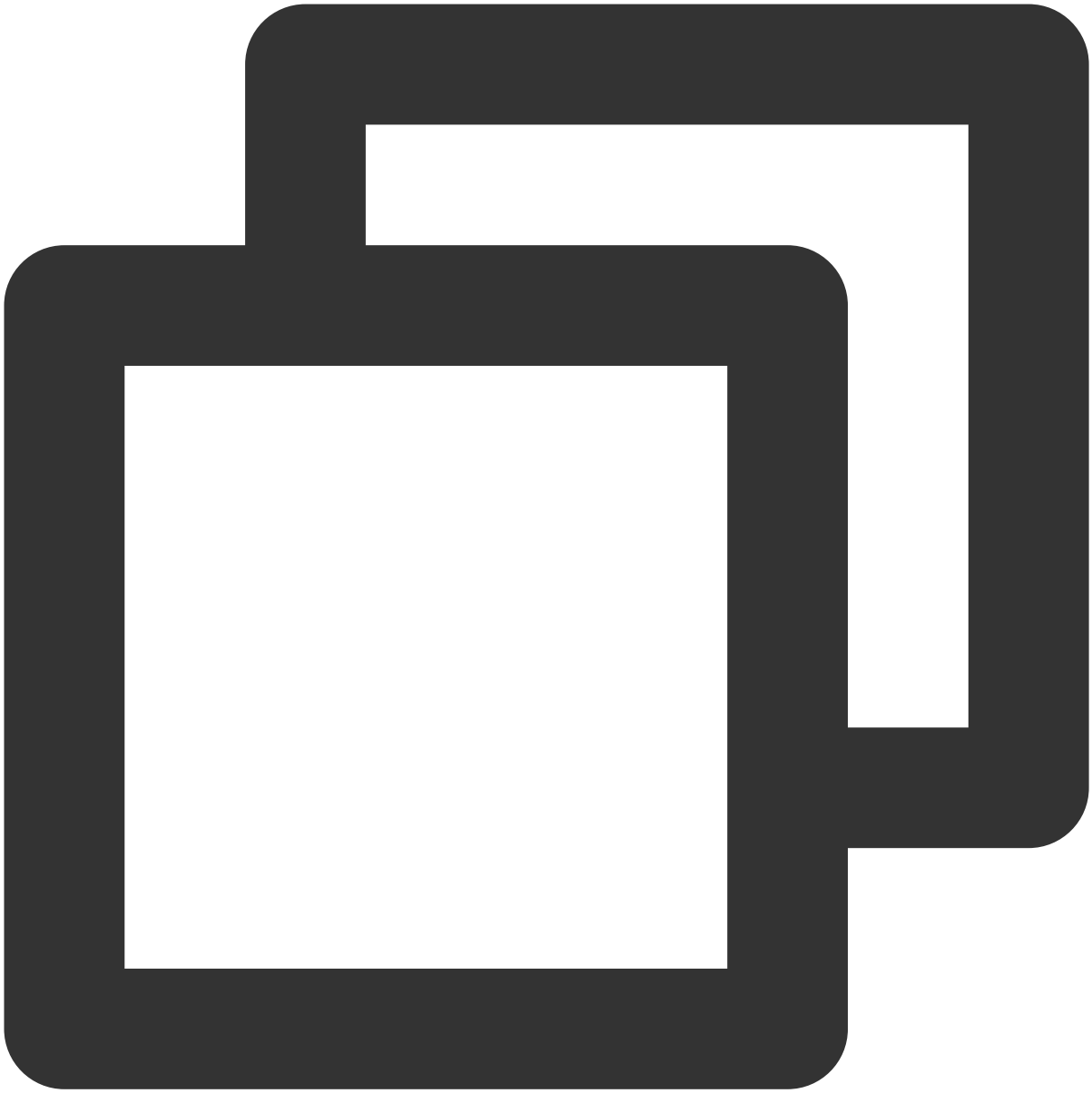
```
SELECT TABLE_NAME, TABLE_TYPE, ENGINE, TABLE_ROWS FROM INFORMATION_SCHEMA.TABLES WH

SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS WHERE ABLE_SCHEMA='db'
```

2. 在结构导出阶段，会有导出线程数（默认8）的连接执行如下 SQL。



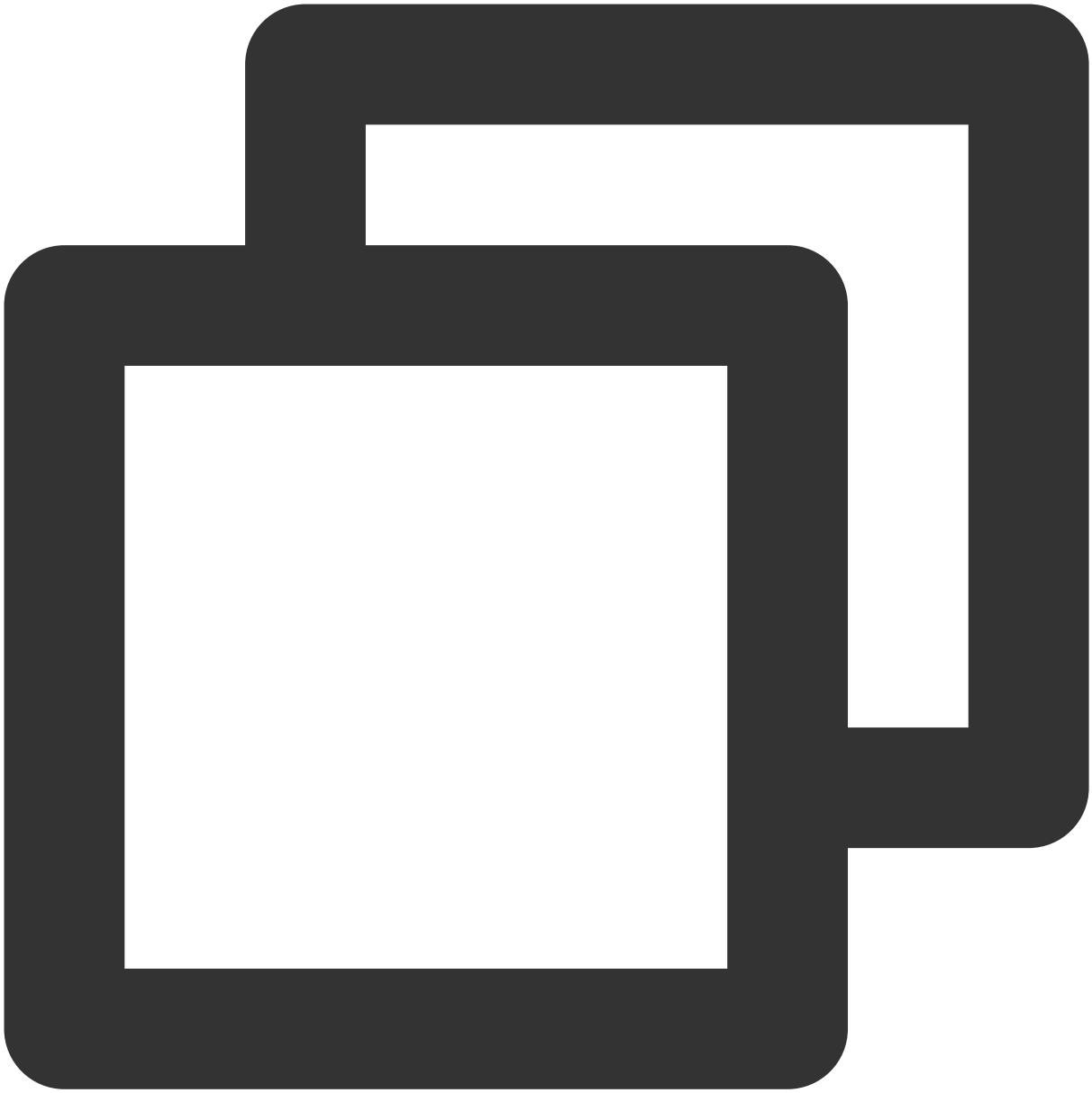
```
SELECT COLUMN_NAME, DATA_TYPE, COLUMN_TYPE, NUMERIC_PRECISION, NUMERIC_SCALE, CHARA
   "where TABLE_SCHEMA='db' and TABLE_NAME='db';
```

3. 数据导出阶段，会有导出线程数（默认8）的连接执行如下 SQL。



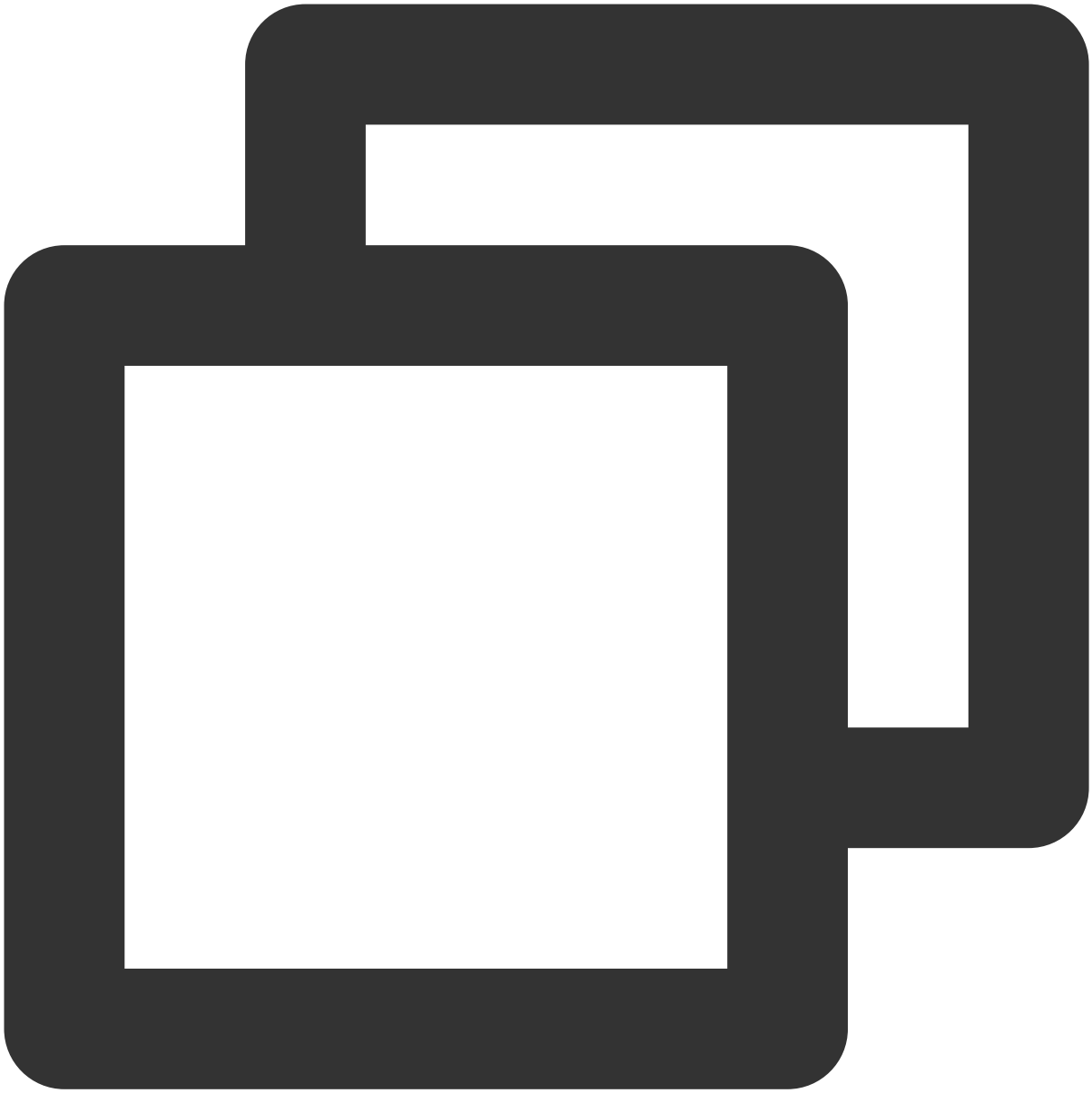
```
SELECT /*!40001 SQL_NO_CACHE */ column FROM db.table where id>= 'id' AND id <
'id';
```

4. 无锁导出时，会有如下类似锁表 SQL，锁表只是为了获取无主键表的一致性位点，获取后即会解锁。



```
lock table xxx read
```

5. 有锁导出时，会有类似如下 SQL 的全局锁。



```
flush table xxx with read lock
```

使用 DTS 进行数据迁移/同步，对目标数据库有啥影响？

全量阶段，DTS 写入目标库时，**对目标库的主要影响在 CPU 和 IOPS。**

源库为 MySQL，规格8核16G，DTS 任务默认采用8线程并发，在网络无瓶颈的情况下，DTS 全量导入阶段对目标库的性能影响：占用目标库 CPU 约20%-49%，占用 IOPS 约1200-3100，占用小于8个活跃 session 连接数。

全量阶段，目标库的连接详情如下：

有小于8个连接在批量创建结构。

有小于8个连接在批量写数据，类似如下语句：

```
insert into xxx (id,name,msg) values (xxx);
```

增量阶段，DTS 会把源数据库 binlog 中的增量数据解析成 SQL，然后在目标数据库中执行，总连接数小于32个 session，其中：

DDL 会串行执行，执行 DDL 时，不会有其他 DML 执行。

DML 最多会有32个连接（都为短连接，超时时间30秒），其中 DML 只是简单的 insert、update、delete、replace 语句。

使用 DTS 进行数据迁移，目标库是否需要为空？

数据迁移到目标库时，可以选择整个实例迁移，也可以选择指定对象迁移。

整个实例迁移：要求目标数据库为空，需要先清空目标库，不为空系统校验不通过，无法发起任务。

指定对象迁移：选择指定的库、表等对象进行迁移，系统会校验源库和目标库是否有同名的库表，如果有，则会校验不通过，提示用户修改后再进行迁移任务。

DTS 是否支持源库使用只读实例进行迁移？

支持。源库为自建数据库的场景，在 DTS 连接源库的配置中，输入只读实例的 IP 即可；源库为腾讯云的云数据库实例的场景，仅订阅支持连接只读实例，需要提交工单申请。

DTS 是否支持源库使用备库进行迁移？

源库为自建数据库的场景，在 DTS 连接源库的配置中，输入备库的 IP 即可；源库为腾讯云数据库实例的场景，在 DTS 连接源库的配置中，只能选择实例 ID，无法连接备库。

数据迁移过程中源库是否支持写入？

支持，迁移过程中源库可以正常进行数据写入，但在结构迁移、全量迁移阶段不能对源库进行 DDL 操作，否则可能会导致任务失败。等到增量迁移阶段，DDL 和 DML 都可以正常操作。

数据迁移过程中目标库是否支持写入？

目标库不需要设置为只读，但是建议不要对目标库进行写入，如果在迁移过程中，用户同时向目标库写入，可能会导致最终源和目标的数据不一致。

DTS 是否支持两端都是线下库的迁移？

不支持，源端或者目标端必须有一个是腾讯云数据库。

DTS 是否支持两个不同腾讯云账号下的 TencentDB 实例之间的数据迁移？

支持，进行跨腾讯云账号下 TencentDB 实例间的迁移，需要以目标 TencentDB 实例所属的腾讯云账号登录 DTS。具体操作请参见 云数据库跨账号实例间迁移。

DTS 是否支持同一个数据库实例内不同库表的迁移？

不支持，DTS 仅支持源库和目标库为不同的数据库实例之间的数据迁移，不支持源库和目标库为同一个数据库实例（同一个数据库实例内不同库表对象的迁移）。

同一个源端数据库是否支持配置多个 DTS 任务，往不同的云数据库实例上迁移？

支持，支持同一个源端迁移到多个目标端（任务可并行处理），也支持多个源端迁移到同一个目标端（需要等前一个任务进行到增量阶段才可发起新一个任务）。需要注意多个任务并行，可能会增加源端或者目标端的访问压力，影响迁移速率。如果您确定需要对同一个源端数据库创建多个任务，可以在创建第一个迁移任务后，通过**操作**列**更多 > 创建类似任务**，快捷创建相同的任务。

DTS 是否支持定时自动迁移？

支持，您可以创建 DTS 后，在修改配置时将选择定时执行的选项，并配置定时迁移时间。

迁移过程中可以监控任务的进度吗？

可以，您可以在腾讯云 DTS 控制台 数据迁移页面查看迁移任务进度。

为什么数据增量迁移服务会有15天设置?

数据增量迁移服务目前使用的是就近代理服务器接入，通过内网专线降低了走公网的抖动问题，确保了数据传输的质量，15天的设置是为了能有效降低代理的服务器的连接压力，如果超过15天，现有阶段并不会进行强制断开，只是从合理使用迁移资源出发做了时间设置规定。

在迁移数据过程中，如何保障数据准确性？

DTS 内部采用腾讯云自研数据迁移架构，对传输链路进行实时的数据准确性校验，快速发现并纠正传输数据，保障传输数据可靠性。

数据校验为什么需要源数据库实例不为只读？

数据校验需要在源实例中创建新库 `__tencentdb__`，并在该库下写入 CheckSum 表，在该实例只读时将会跳过数据校验阶段。

DTS 数据迁移能指定库表进行迁移么？

可以，迁移对象可以选择整个实例，也可以选择指定库表对象。

数据迁移什么时候结束？

用户选择增量迁移时，任务长时间没有结束，有可能需要用户自己进行结束操作。
如果迁移类型选择**结构迁移**或者**全量迁移**，则任务完成后会自动结束，不需要用户手动结束。
如果迁移类型选择**全量 + 增量迁移**，则全量迁移完成后会自动进入增量数据同步阶段，增量数据同步不会自动结束，需要您手动单击完成结束增量数据同步。
请选择合适时间手动完成增量数据同步，并完成业务切换。
观察迁移阶段为增量同步，并显示无延迟状态，将源库停写几分钟。

目标与源库数据差距为0MB及目标与源库时间延迟为0秒时，手动完成增量同步。

为什么全量迁移前后数据的大小不一致？

因为源库和目标库的碎片空间不一样，源库可能存在一些空洞数据，因此全量迁移完成后目标库的表存储空间很可能会比源库的表存储空间小。建议用户迁移完成后使用 数据一致性校验 来核对源库和目标库内容是否一致。

DTS 数据传输服务，是否支持跨国数据库迁移？

支持，接入方式选择公网可以实现跨国数据传输。

DTS 迁移过程中任务异常中断，是否可以重新启动？

可以，用户在配置任务时，可以设置自动重试策略，在任务发生异常中断（如源库或者目标库短暂不可用，网络问题等），DTS 支持在设置的时间范围内进行自动重试，无需手动干预。

MySQL 常见问题

MySQL 迁移过程中是否会加锁？

MySQL 迁移过程中的加锁，指的是对源数据库加全局锁（FTWRL），选择全量数据迁移的场景中，才会涉及到加锁处理。
当前 DTS 在 MySQL/MariaDB//Percona/TDSQL-C MySQL/TDSQL MySQL/TDSQL TDStore 之间的数据迁移中，默认使用无锁迁移。无锁迁移指对源库不加全局锁（FTWRL），仅对无主键的表加表锁。

是否支持迁移无主键的表？

支持，但是建议用户选择有主键的表，无主键的表也可以发起任务，但是会造成以下影响：
迁移/同步无主键表，可能会导致数据重复。
无主键表在有 DML 操作时，容易导致数据同步延迟。
进行数据一致性校验的时候，无主键表将被跳过，不支持校验。
源数据库为阿里云 RDS，PolarDB 时，由于 RDS，PolarDB 在 Binlog 中为无主键或无非空唯一键的表加上附加主键列，但在表结构中不可见，可能会导致 DTS 无法识别，数据结果异常。

是否支持腾讯云 MySQL 单节点（原基础版）实例的迁移？

腾讯云 MySQL 单节点（原基础版）作为源可以使用公网模式迁移，不能使用内网模式。
暂不支持将腾讯云 MySQL 单节点（原基础版）作为迁移的目标实例。

对源库设置 binlog_format 为 row 之后，如何确保源库 binlog_format 格式立刻生效？

设置 binlog_format 为 row 后，需重置当前数据库上的所有业务连接（当源库在从机时，还需重置主从同步 SQL 线程），避免当前业务连接继续使用老格式写入。
在上述操作未结束之前，请不要创建或者启动迁移任务，避免产生数据不一致。

如果迁移的源实例里面有 Toku 引擎，迁移会有什么需要注意的？

如果源实例涉及了 Toku 引擎，我们会在迁移时默认转成 InnoDB，然后 Cluster Index 和 TokuDB 的压缩特性的表需要提前做处理，目前迁移不支持，另外对 Toku 引擎的 DDL 操作也是迁移不支持。

MySQL 数据迁移可以迁移用户权限吗？

支持，新版本 DTS 支持对用户的权限进行迁移，具体操作请参考 账号迁移。

数据同步

最近更新时间：2024-07-08 15:38:33

数据同步对目标库有啥影响，是否要求目标库为空？

使用 DTS 进行数据同步对目标库没有影响，不需要目标库为空。DTS 支持检测源库和目标库是否有同名对象，并支持在发生同名对象冲突时按照设置的策略进行处理，将任务报错提醒用户，或者忽略报错继续任务。

源/目标实例发生 HA（High Availability）切换时，同步服务是否会受影响？

源实例支持并开启 GTID（Global Transaction Identifier）时，在增量同步阶段，如果源实例发生 HA，那么服务会自动重连，同步数据流在 HA 完成后迅速恢复。
目标实例在增量同步阶段发生目标实例 HA，服务也会自动重连，同步数据流在 HA 完成后迅速恢复。

支持将高版本实例的数据同步到低版本的实例吗？

不支持。同一个类型的数据库，目标实例的大版本必须不小于源实例。以 MySQL 实例为例，不支持 MySQL 5.7 的实例同步到 MySQL 5.6。

源/目标能否为云下实例？

可以。可以将云上的数据库同步到云下，接入方式支持公网、云联网等。

在双向同步拓扑中，是否可以对 DDL 进行双向同步？

不可以。创建同步实例时，只能允许其中一个实例进行 DDL 同步，否则检测算法检测到 DDL 循环，会禁止其中一个实例的创建。

是否支持非事务引擎？

当前技术方案采用在事务中打上路由信息来标记事务的来源，依赖于事务的原子性。基于非事务引擎的库表，会破坏了事务的原子性，无法保证数据一致，不建议用户使用。

同步任务启动后，是否支持追加同步对象或者删除已勾选的同步对象？

支持。选择需要修改的同步任务，通过**操作 > 修改同步配置**，可以修改同步任务配置。
可支持增加/删除同步对象、修改主键冲突策略、SQL 同步策略等。修改同步配置时，已有的同步任务不会暂停，也不会受影响。更多详情请参考 修改同步配置。

数据订阅 Kafka 版常见问题

最近更新时间：2024-07-08 15:39:01

为什么我消费不到数据？

排查网络问题，Kafka 服务器地址为腾讯云内网地址，只能在腾讯云与订阅实例相同地域的私有网络内访问。
排查订阅 topic、内网地址、消费组名称、帐号、密码等是否正确。可在 数据订阅控制台 单击订阅名进入订阅详情页和消费管理页查看。
排查加密参数是否正确，参见 Kafka 使用何种认证机制。

数据格式是怎么样的?

数据订阅 Kafka 版使用 Protobuf 进行序列化。Protobuf 协议文件可在 此处 下载，Demo工 程中也包含了协议文件，具体请参考 生产和消费逻辑讲解。

Kafka 使用何种认证机制？

如下所示：

```
def block_consume(brokers, topic, group, user, password, trans2sql):
    # create a kafka consumer
    consumer = kafka.KafkaConsumer(topic,
                                   auto_offset_reset='earliest',
                                   enable_auto_commit=False,
                                   group_id=group,
                                   sasl_plain_username=user,
                                   sasl_plain_password=password,
                                   sasl_mechanism='SCRAM-SHA-512',
                                   security_protocol='SASL_PLAINTEXT',
                                   bootstrap_servers=brokers)
```

何时进行 Kafka commit?

首先请将 Kafka 的 enable_auto_commit 参数设置为 false，以关闭自动 commit。生产者会在消息序列中的合适位置插入 Checkpoint 消息，消费者消费到 Checkpoint 消息后进行 commit，这样有利于保证消息的完整性。

服务端消息保留多久，如何设置消费 offset？

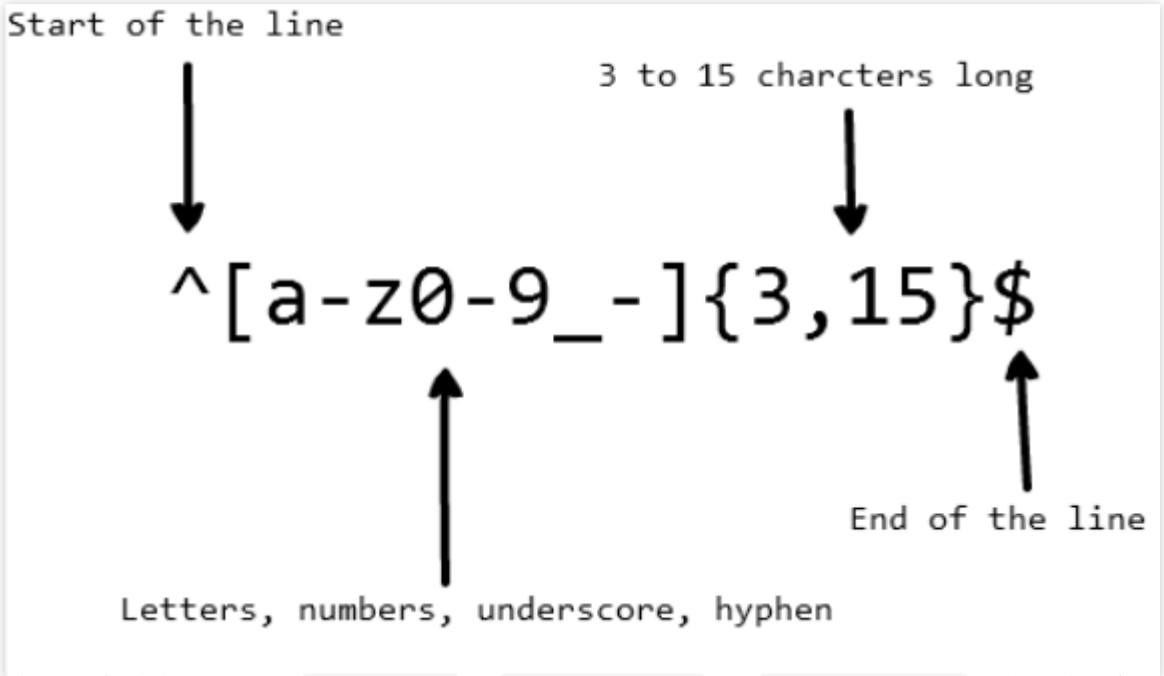
服务端消息保留1天。可以根据需要配置 Kafka 的 auto_offset_reset 参数为 earliest 或者 latest。如果需要从具体的 offset 开始消费，则可以利用 Kafka Client 提供的 seek 功能，重新设置消费 offset。

数据订阅正则表达式

最近更新时间：2024-07-08 15:39:41

什么是正则表达式 ？

正则表达式是一种被用于从文本中检索符合某些特定模式的文本。
正则表达式是从左到右来匹配一个字符串的。"Regular Expression" 这个词太长了，我们通常使用它的缩写 "regex" 或者 "regexp"。
正则表达式可以被用来替换字符串中的文本、验证表单、基于模式匹配从一个字符串中提取字符串等。
当您正在编写应用程序，并且希望在用户选择用户名时设置规则。我们希望用户名可以包含字母，数字，下划线和连字符。
为让其展示美观，想要限制用户名中的字符数量。可以使用以下正则表达式来验证用户名：

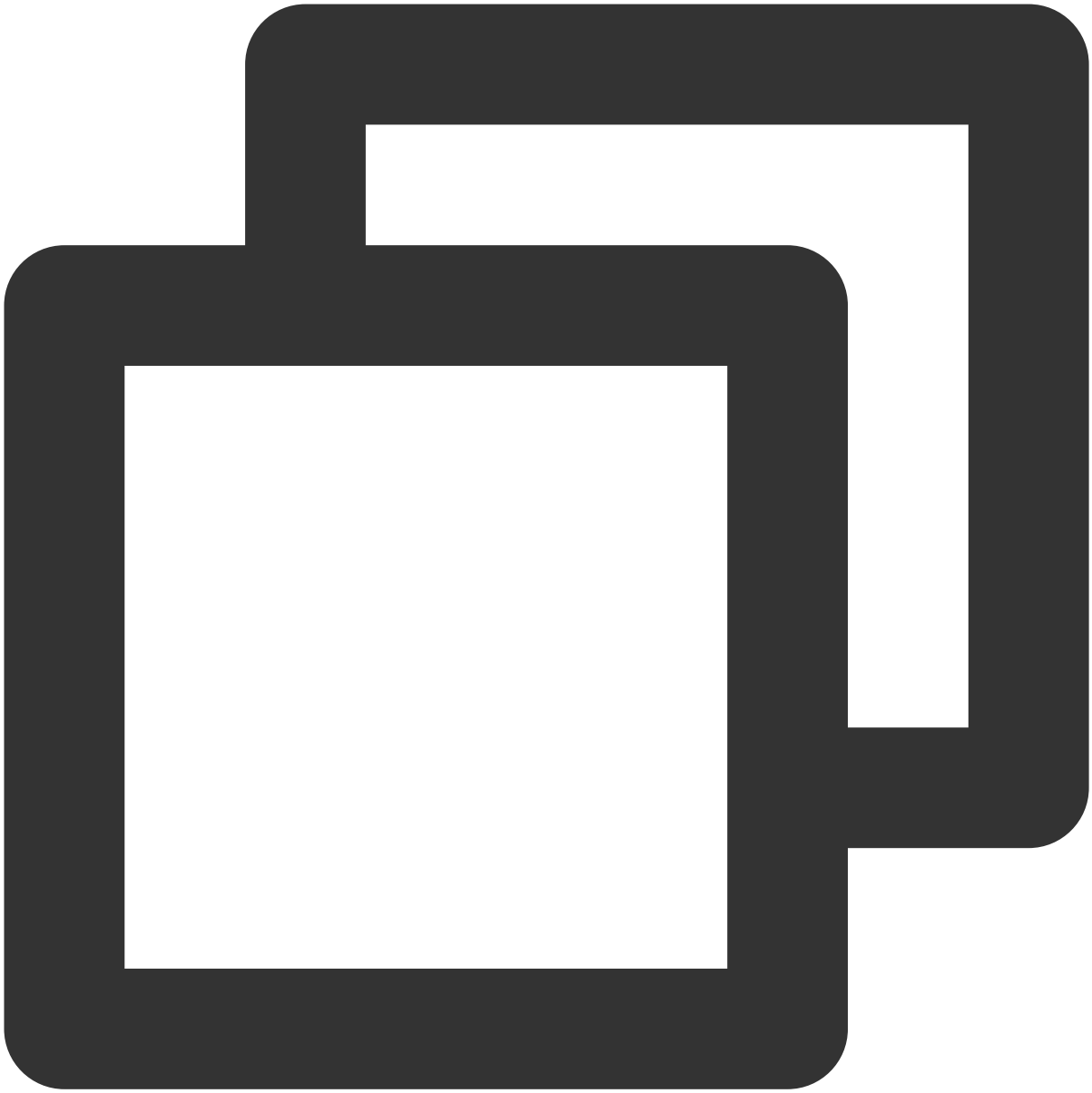


上面这个正则表达式可以匹配 `john_doe` ， `jo-hn\\_doe` 和 `john12\\_as` 。但是它不能匹配 `Jo` ，因为该字符串里面包含了大写字符，并且它太短了。

目录

基本匹配

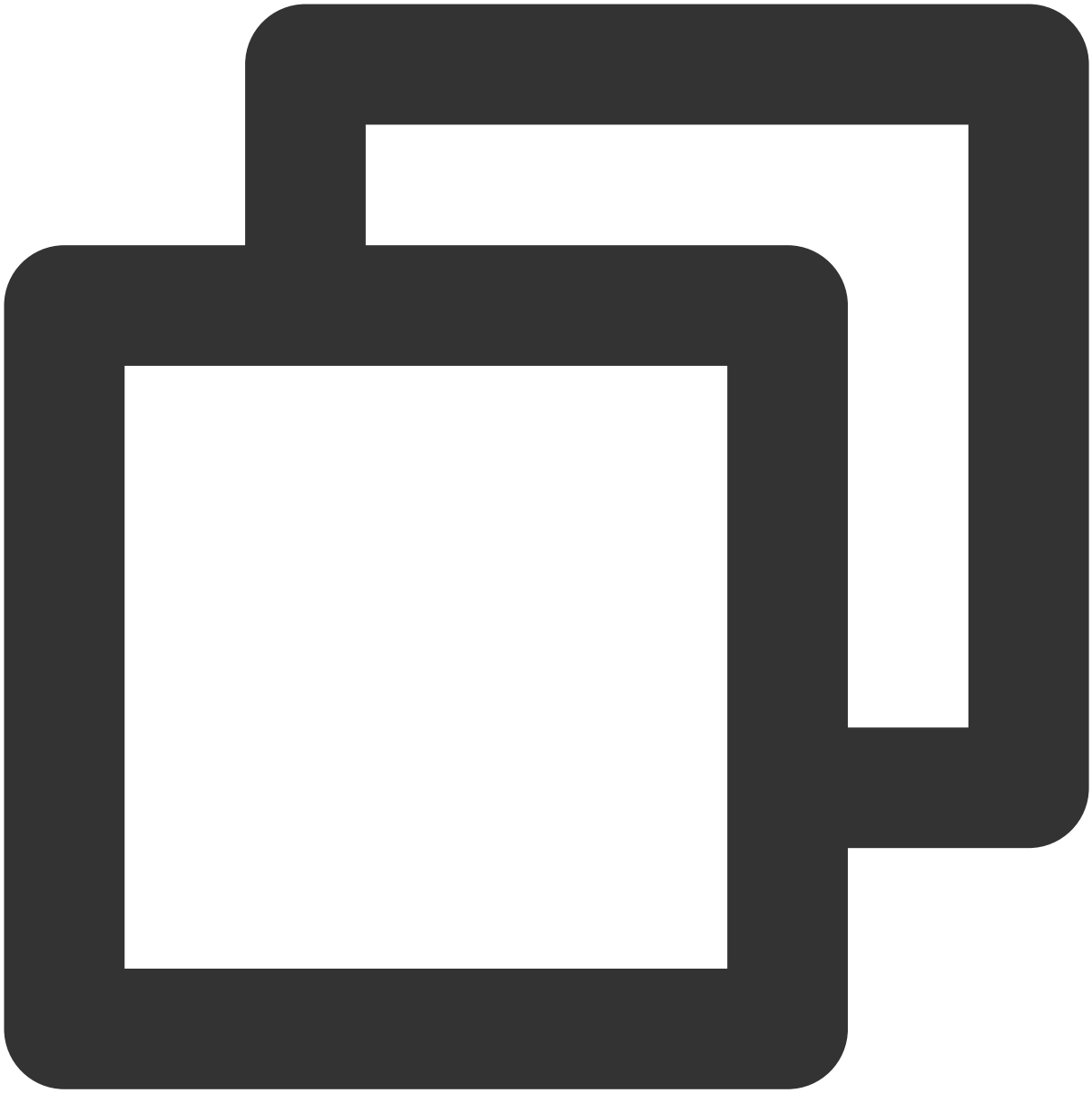
正则表达式只是我们用于在文本中检索字母和数字的模式。例如正则表达式 `cat`，表示: 字母 `c` 后面跟着一个字母 `a`，再后面跟着一个字母 `t`。



```
"cat" => The cat sat on the mat
```

正则表达式 `123` 会匹配字符串 "123"。通过将正则表达式中的每个字符逐个与要匹配的字符串中的每个字符进行比较，来完成正则匹配。

正则表达式通常区分大小写，因此正则表达式 `Cat` 与字符串 "cat" 不匹配。



```
"Cat" => The cat sat on the Cat
```

元字符

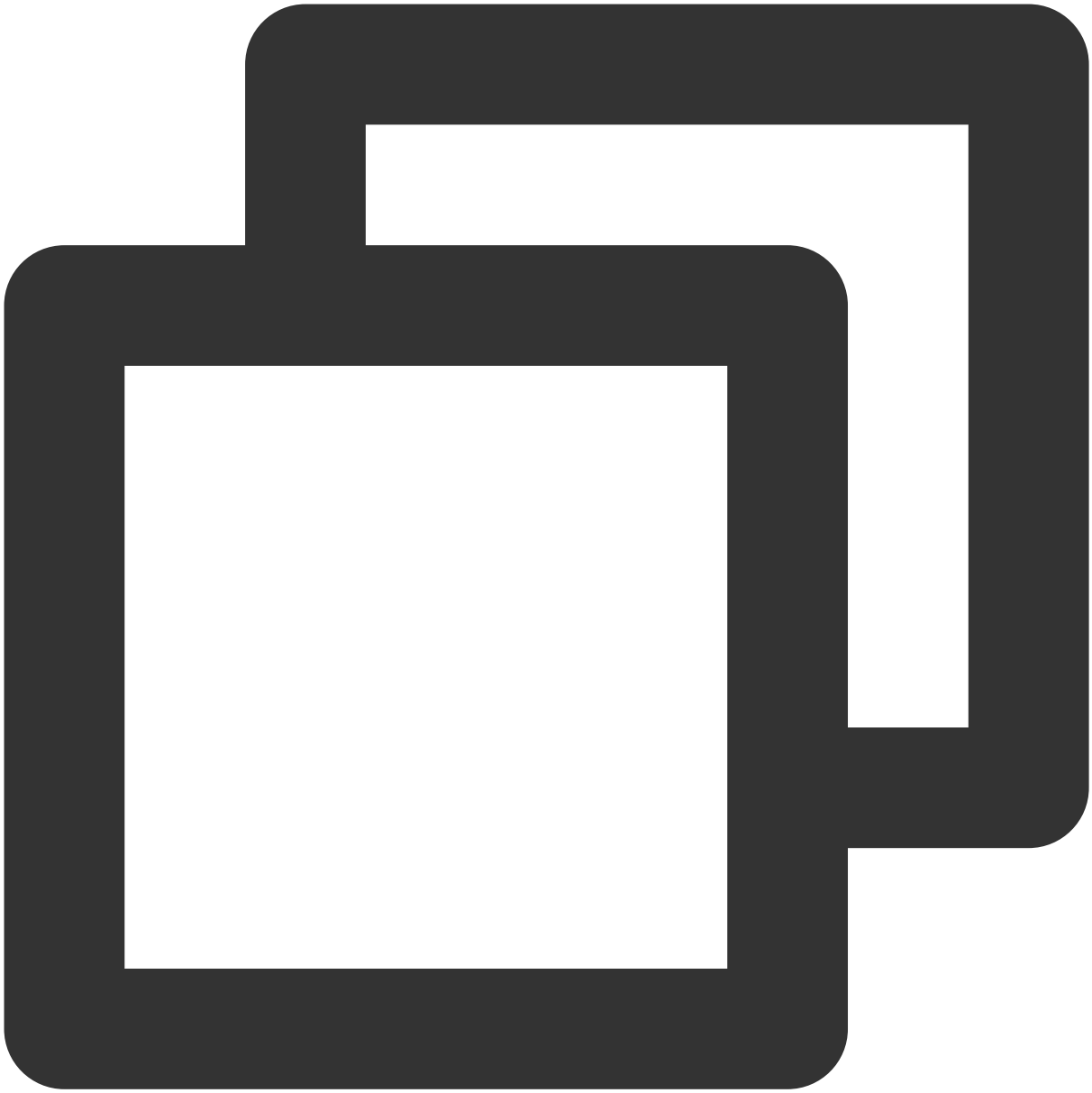
元字符是正则表达式的基本组成元素。元字符在这里跟它通常表达的意思不一样，而是以某种特殊的含义去解释。有些元字符写在方括号内的时候有特殊含义。

元字符如下:

元字符	描述
.	匹配除换行符以外的任意字符。
[ ]	字符类，匹配方括号中包含的任意字符。
[^]	否定字符类。匹配方括号中不包含的任意字符。
*	匹配前面的子表达式零次或多次。
+	匹配前面的子表达式一次或多次。
?	匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。
{n,m}	花括号，匹配前面字符至少 n 次，但是不超过 m 次。
(xyz)	字符组，按照确切的顺序匹配字符xyz。
|	分支结构，匹配符号之前的字符或后面的字符。
&#92;	转义符，它可以还原元字符原来的含义，允许您匹配保留字符 `[ ] ( ) { } . * + ? ^ $ \\ |` 。
^	匹配行的开始。
$	匹配行的结束。

英文句号

英文句号 `.` 是元字符的最简单的例子。元字符 `.` 可以匹配任意单个字符。它不会匹配换行符和新行的字符。例如正则表达式 `.ar` ，表示: 任意字符后面跟着一个字母 `a` ，再后面跟着一个字母 `r` 。

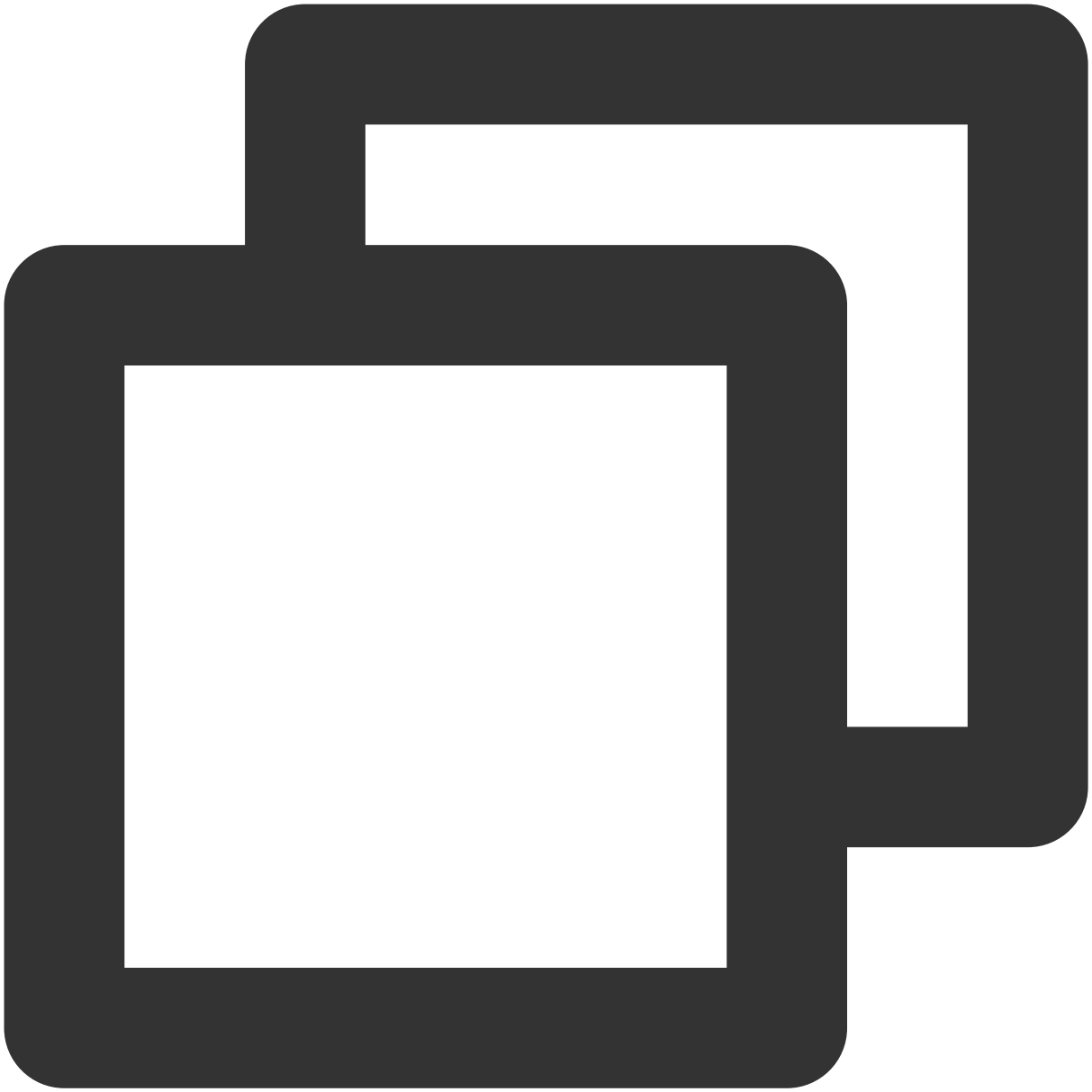


```
".ar" => The car parked in the garage.
```

字符集

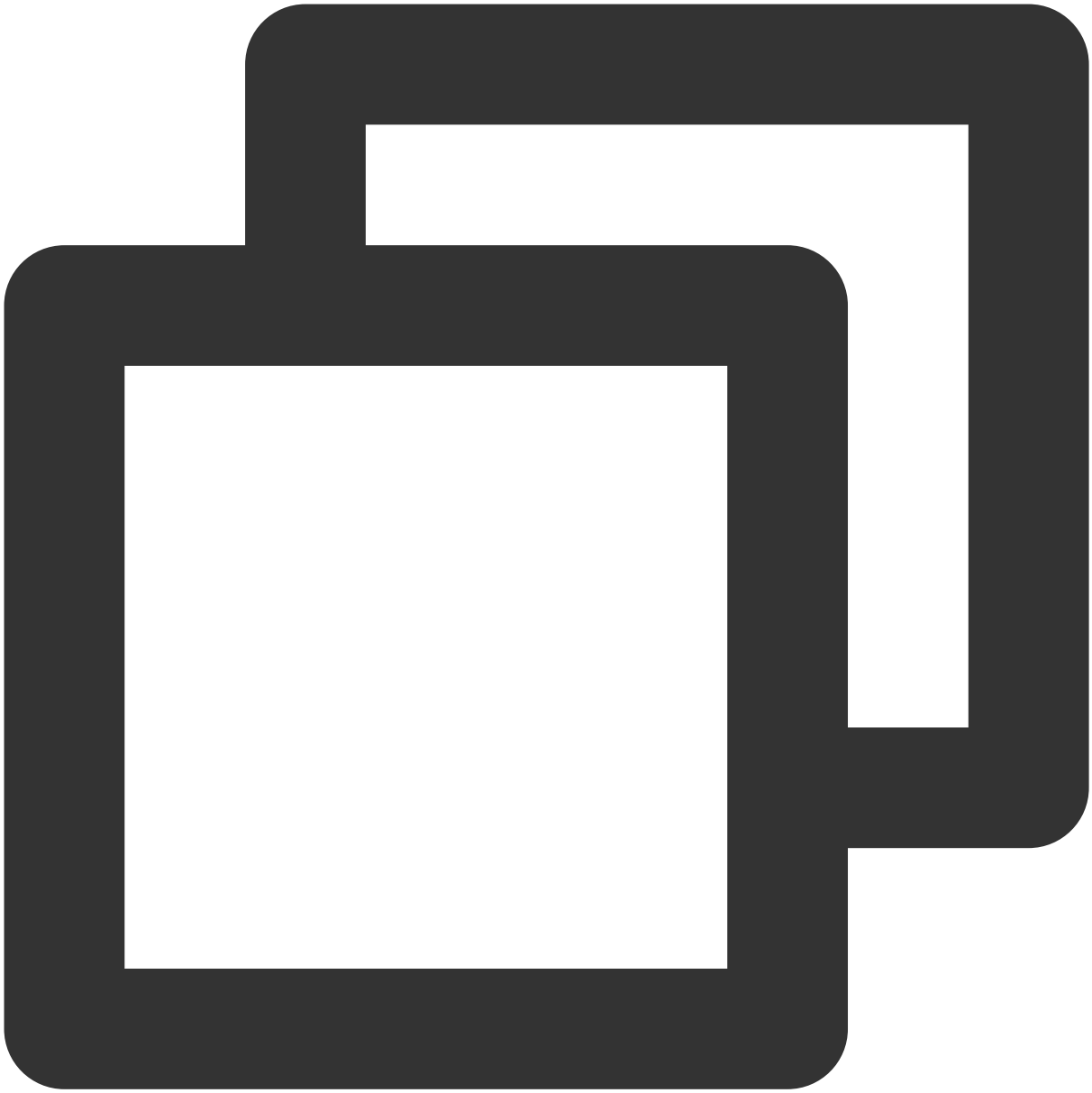
字符集也称为字符类。方括号被用于指定字符集。使用字符集内的连字符来指定字符范围。方括号内的字符范围的顺序并不重要。

例如正则表达式 `[Tt]he` ，表示: 大写 `T` 或小写 `t` ，后跟字母 `h` ，再后跟字母 `e` 。



```
"[Tt]he" => The car parked in the garage.
```

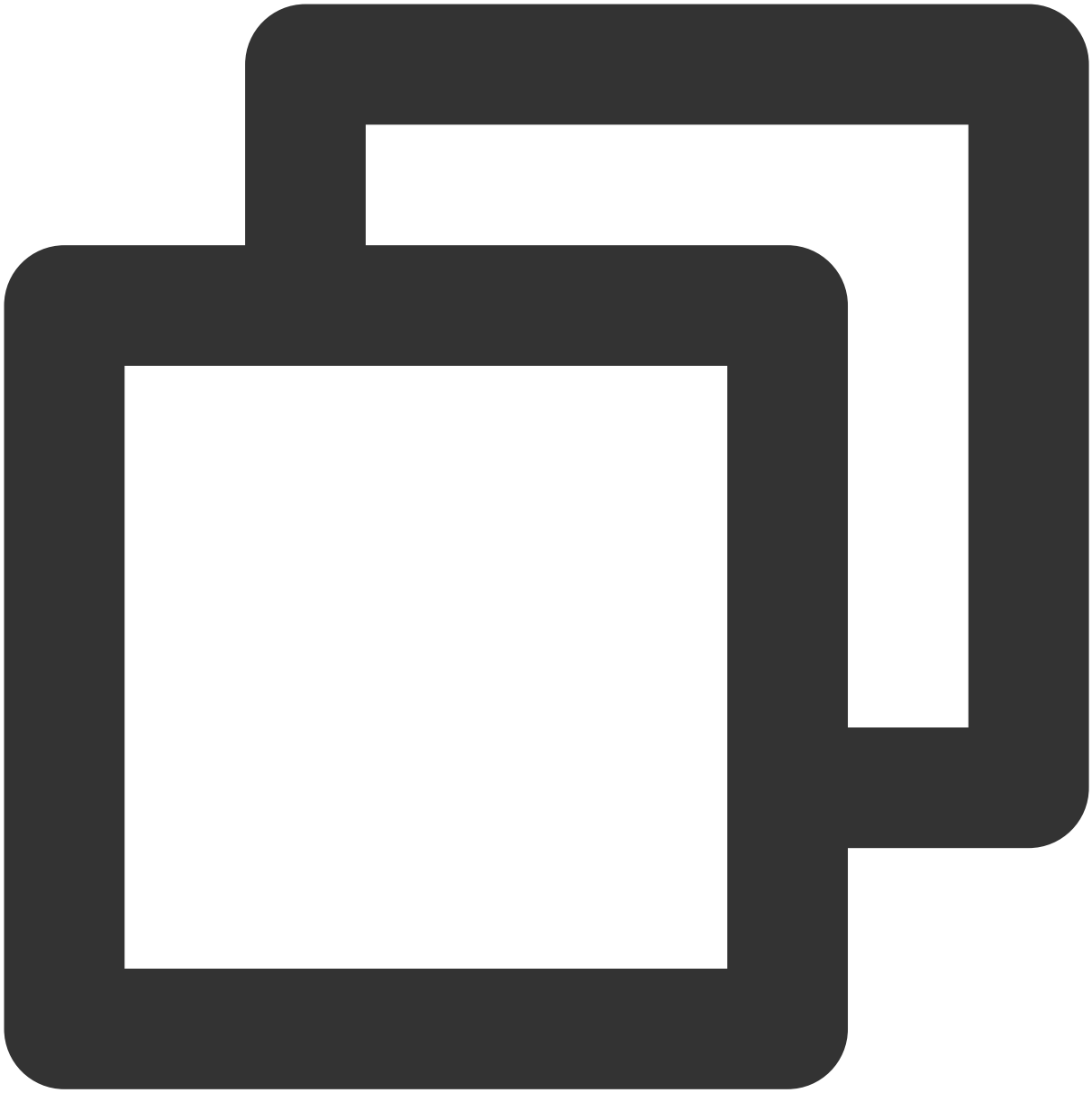
然而，字符集中的英文句号表示它字面的含义。正则表达式 `ar[.]` ，表示小写字母 `a` ，后面跟着一个字母 `r` ，再后面跟着一个英文句号 `.` 字符。



```
"ar[.]" => A garage is a good place to park a car.
```

否定字符集

一般来说插入字符 `^` 表示一个字符串的开始，但是当它在方括号内出现时，它会取消字符集。例如正则表达式 `[^c]ar` ，表示: 除了字母 `c` 以外的任意字符，后面跟着字符 `a` ，再后面跟着一个字母 `r` 。



```
"[^c]ar" => The car parked in the garage.
```

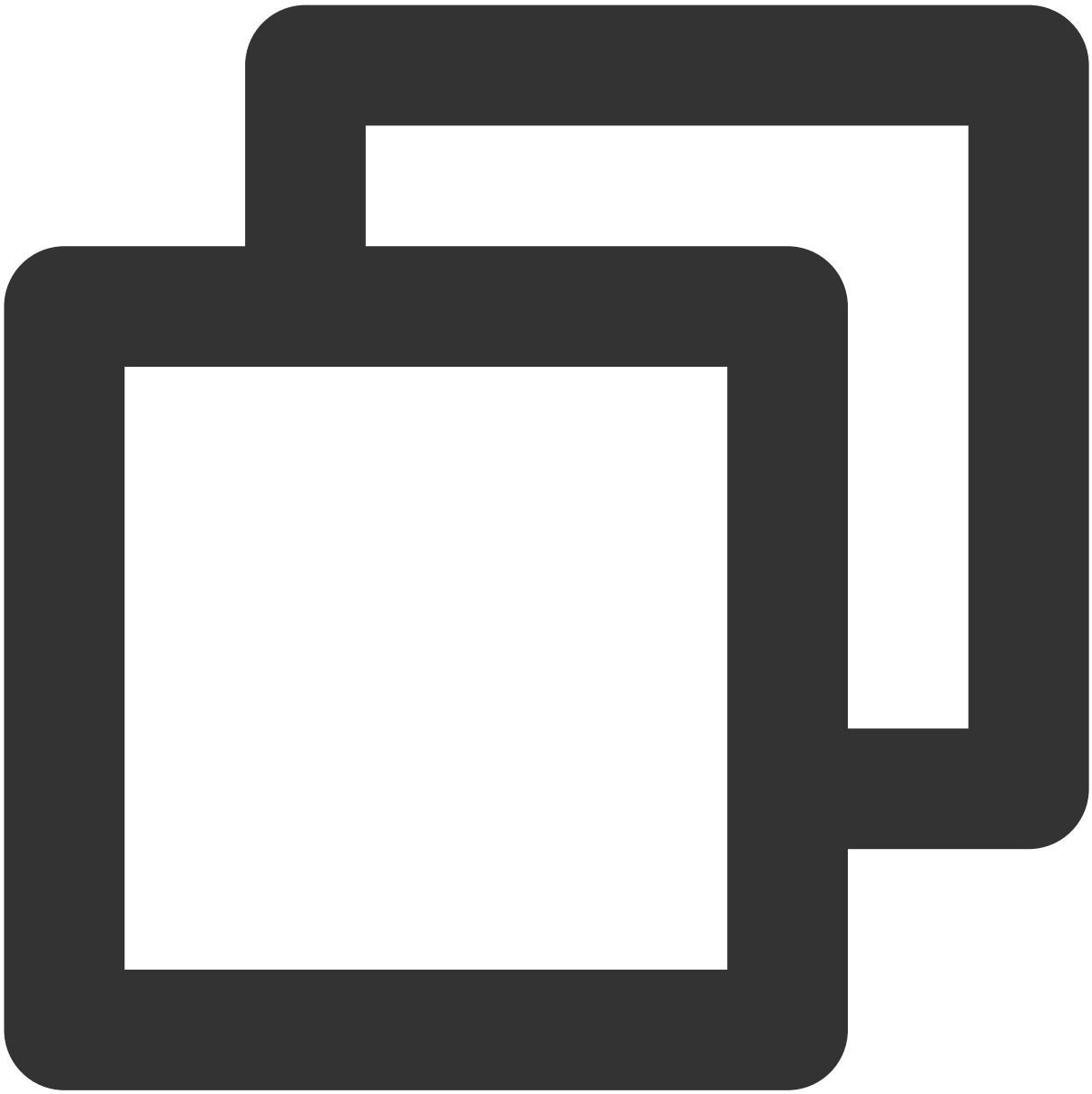
重复

以下元字符 `+`， `*` 或 `?` 用于指定子模式可以出现多少次。这些元字符在不同情况下的作用不同。

星号

该符号 `*` 表示匹配上一个匹配规则的零次或多次。正则表达式 `a*` 表示小写字母 `a` 可以重复零次或者多次。但是它如果出现在字符集或者字符类之后，它表示整个字符集的重复。

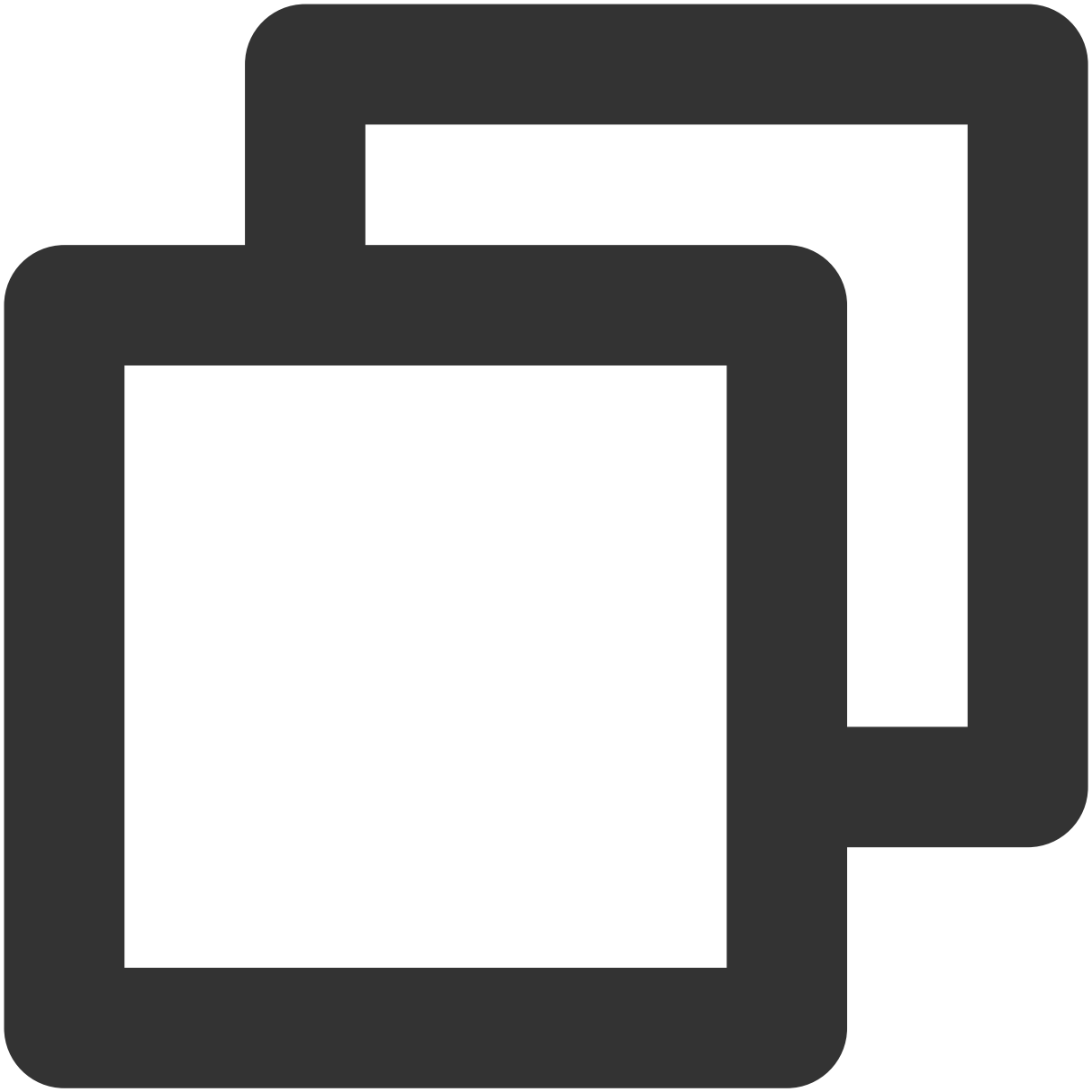
例如正则表达式 `[a-z]*`，表示: 一行中可以包含任意数量的小写字母。



```
"[a-z]*" => The car parked in the garage #21.
```

该 `*` 符号可以与元符号 `.` 用在一起，用来匹配任意字符串 `.*`。该 `*` 符号可以与空格符 `\\s` 一起使用，用来匹配一串空格字符。

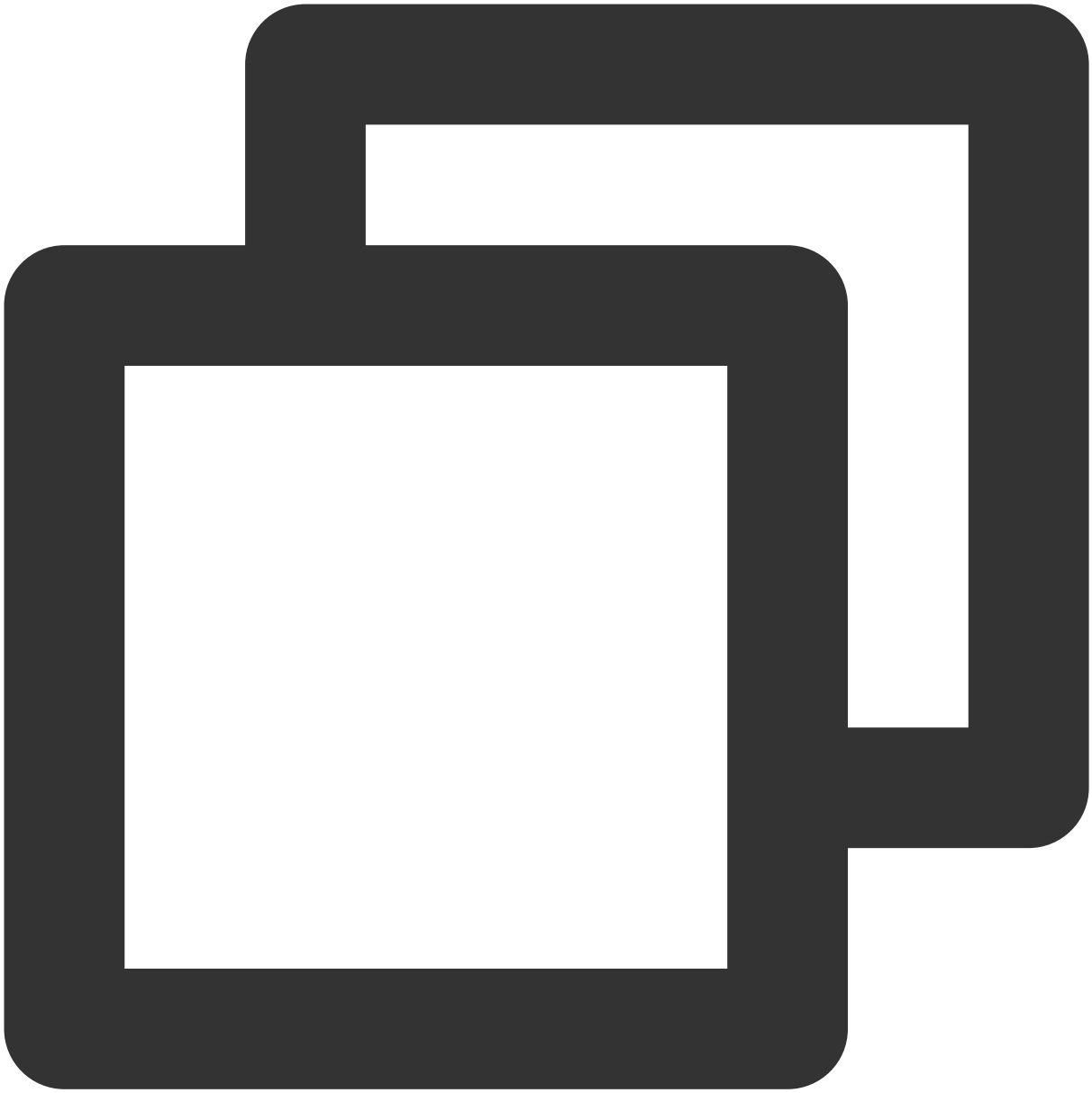
例如正则表达式 `\\s*cat\\s*`，表示: 零个或多个空格，后面跟小写字母 `c`，再后面跟小写字母 `a`，再往后面跟小写字母 `t`，后面再跟零个或多个空格。



```
"\\s*cat\\s*" => The fat cat sat on the cat.
```

加号

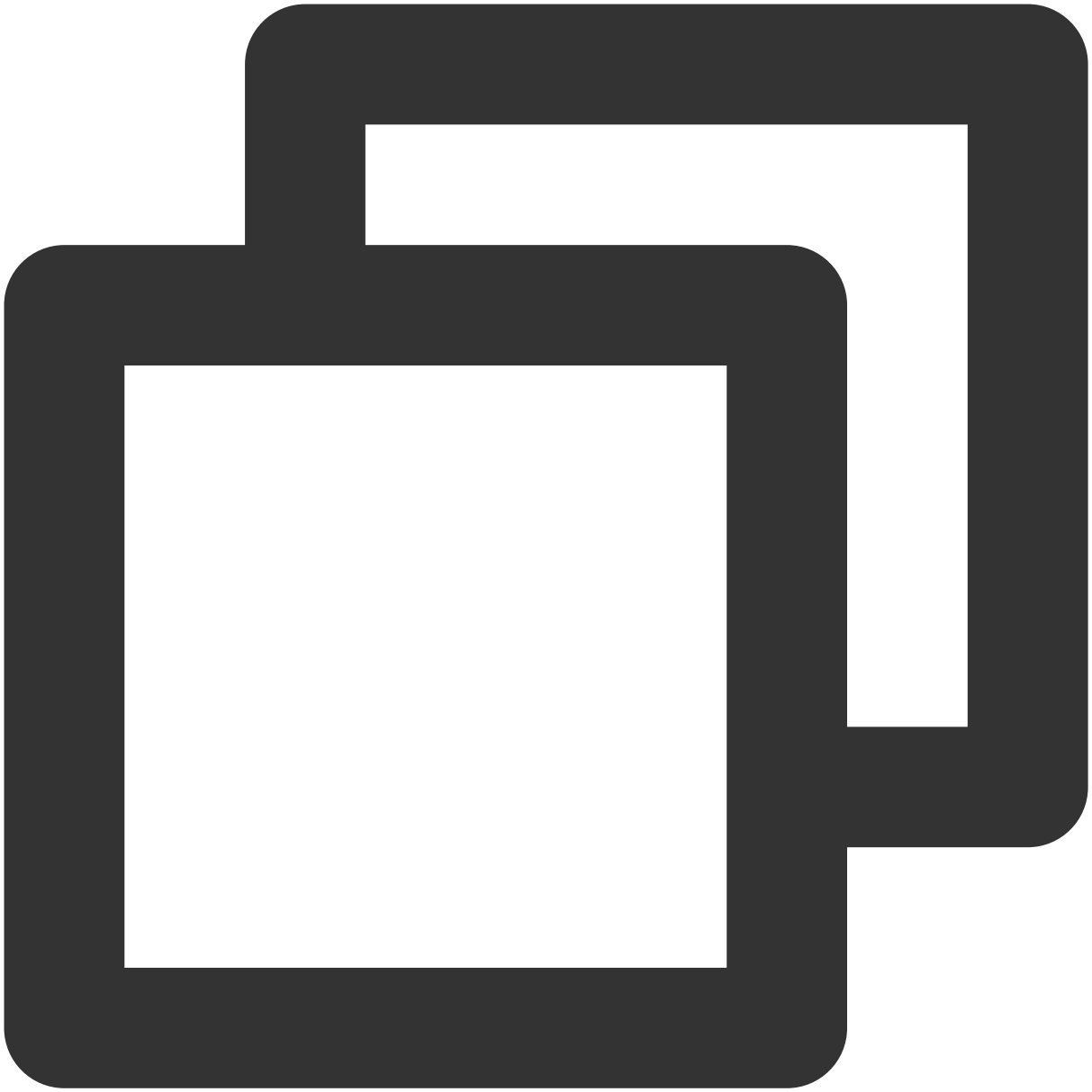
该符号 `+` 匹配上一个字符的一次或多次。例如正则表达式 `c.+t` ，表示: 一个小写字母 `c` ，后跟任意数量的字符，后跟小写字母 `t` 。



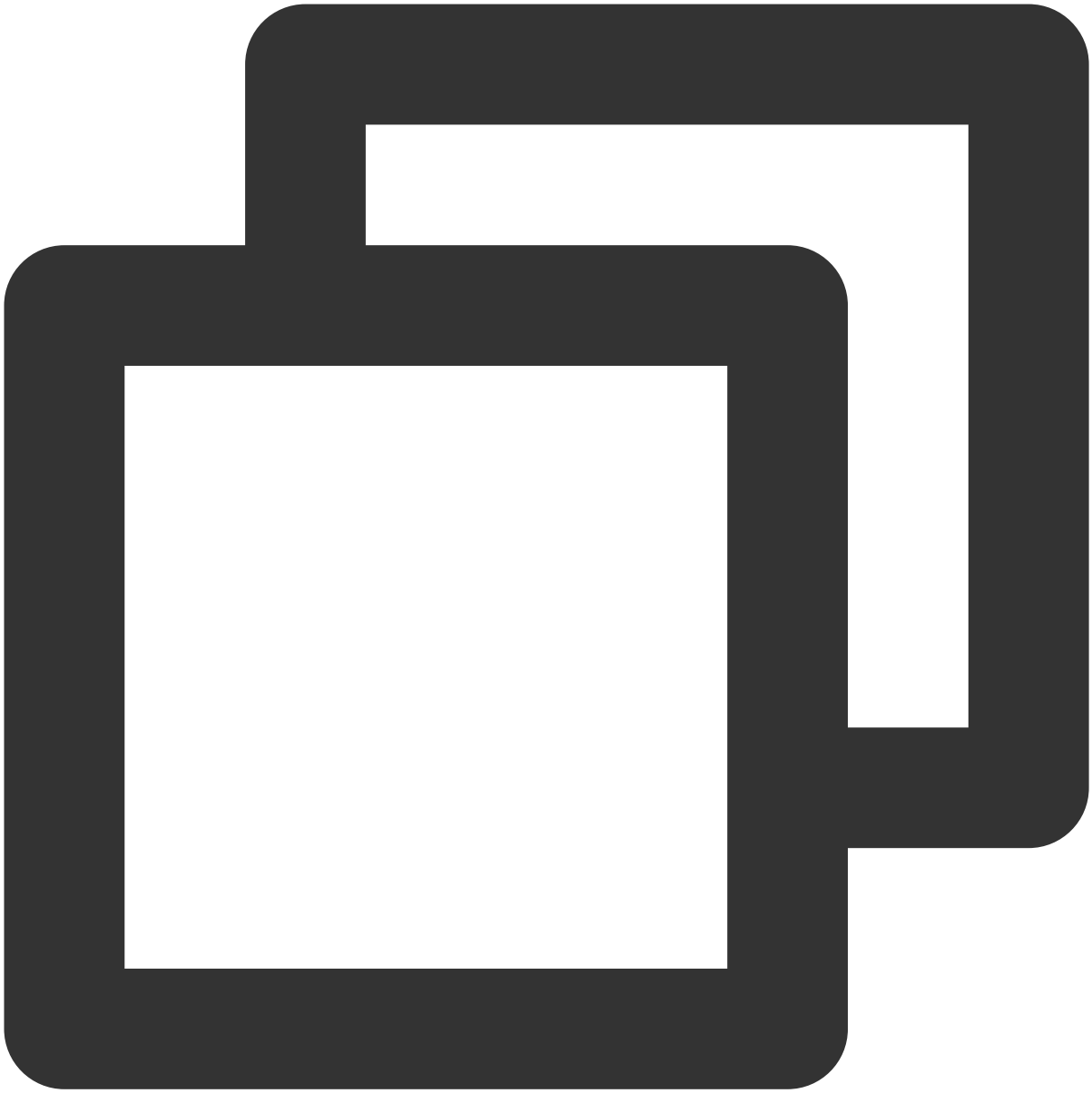
```
"c.+t" => The fat cat sat on the mat.
```

问号

在正则表达式中，元字符 `?` 用来表示前一个字符是可选的。该符号匹配前一个字符的零次或一次。
例如正则表达式 `[T]?he` ，表示: 可选的大写字母 `T` ，后面跟小写字母 `h` ，后跟小写字母 `e` 。



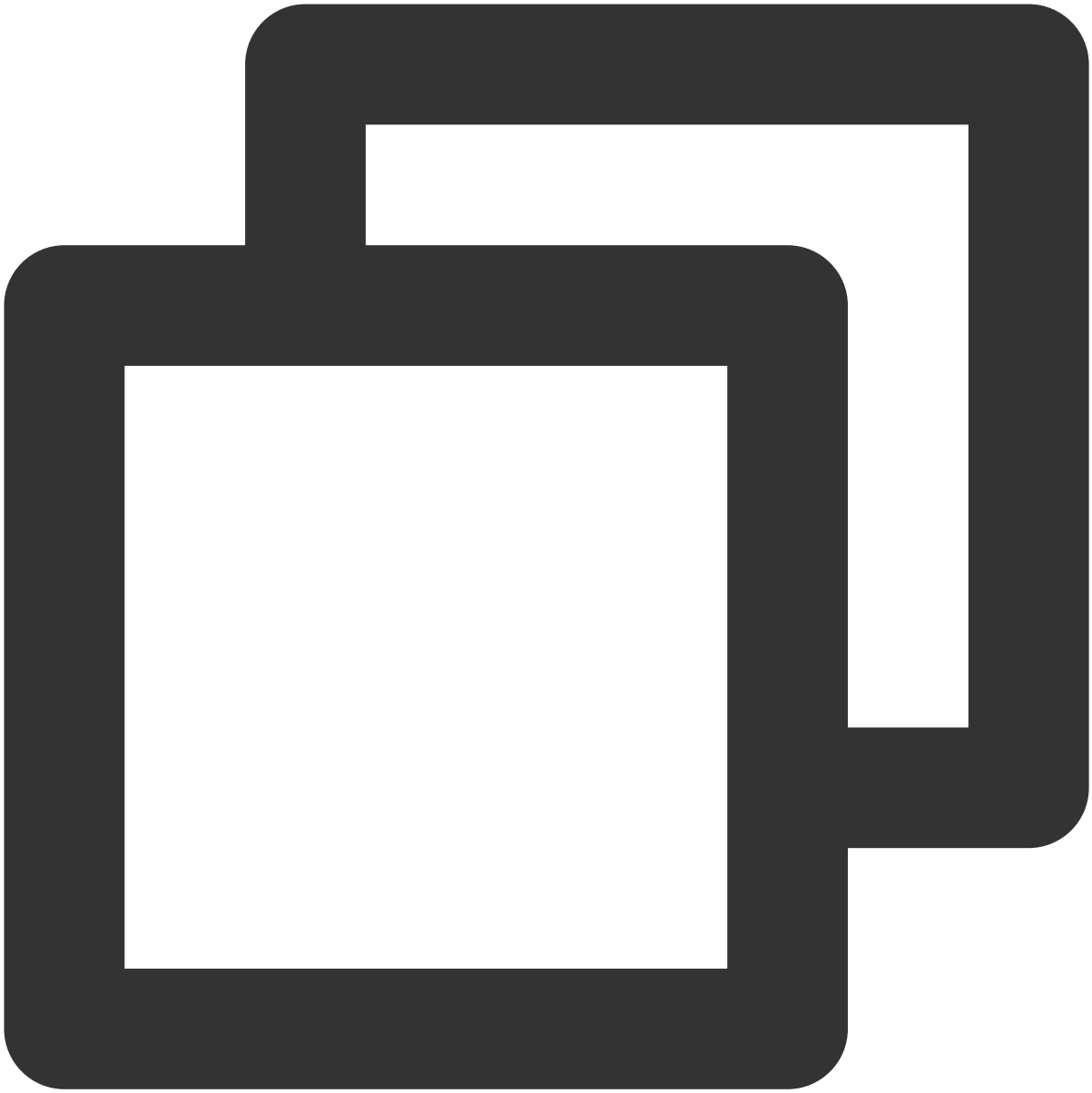
```
"[T]he" => The car is parked in the garage.
```



```
"[T]?he" => The car is parked in the garage.
```

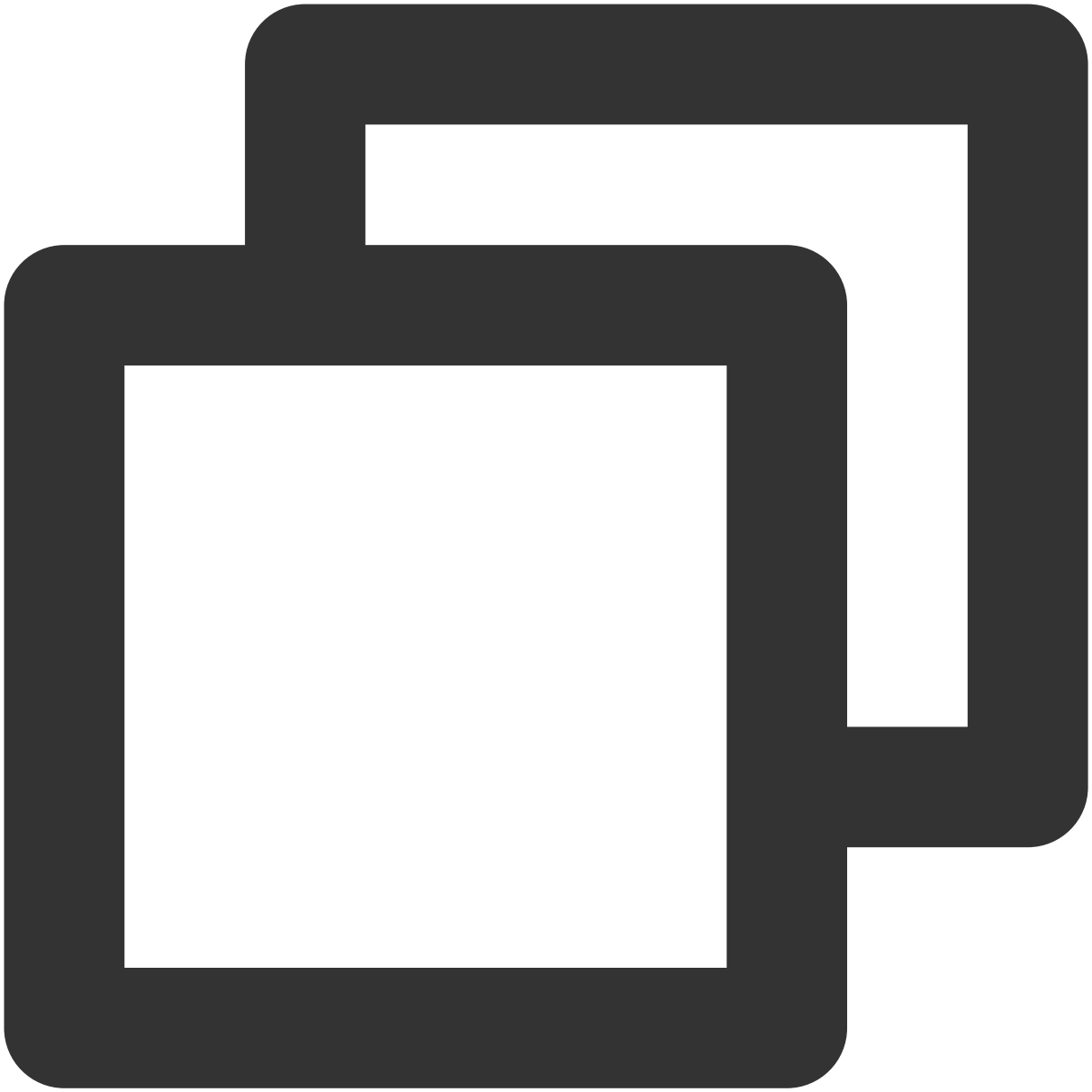
花括号

在正则表达式中花括号(也被称为量词 ?)用于指定字符或一组字符可以重复的次数。例如正则表达式 `[0-9]{2,3}` ，表示: 匹配至少2位数字但不超过3位(0到9范围内的字符)。

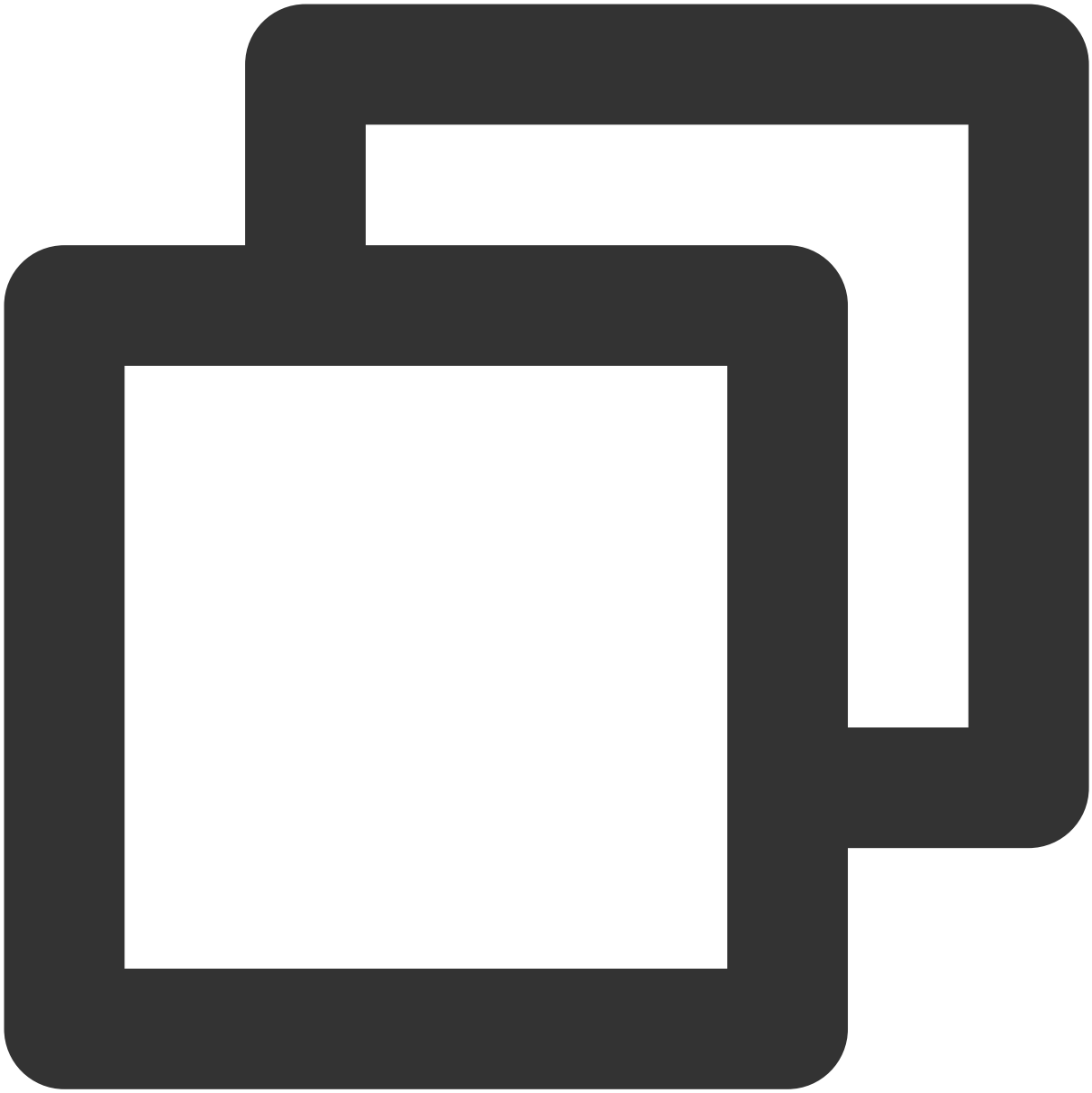


```
"[0-9]{2,3}" => The number was 9.9997 but we rounded it off to 10.0.
```

我们可以省略第二个数字。例如正则表达式 `[0-9]{2,}` ，表示: 匹配2个或更多个数字。如果我们也删除逗号，则正则表达式 `[0-9]{2}` ，表示: 匹配正好为2位数的数字。



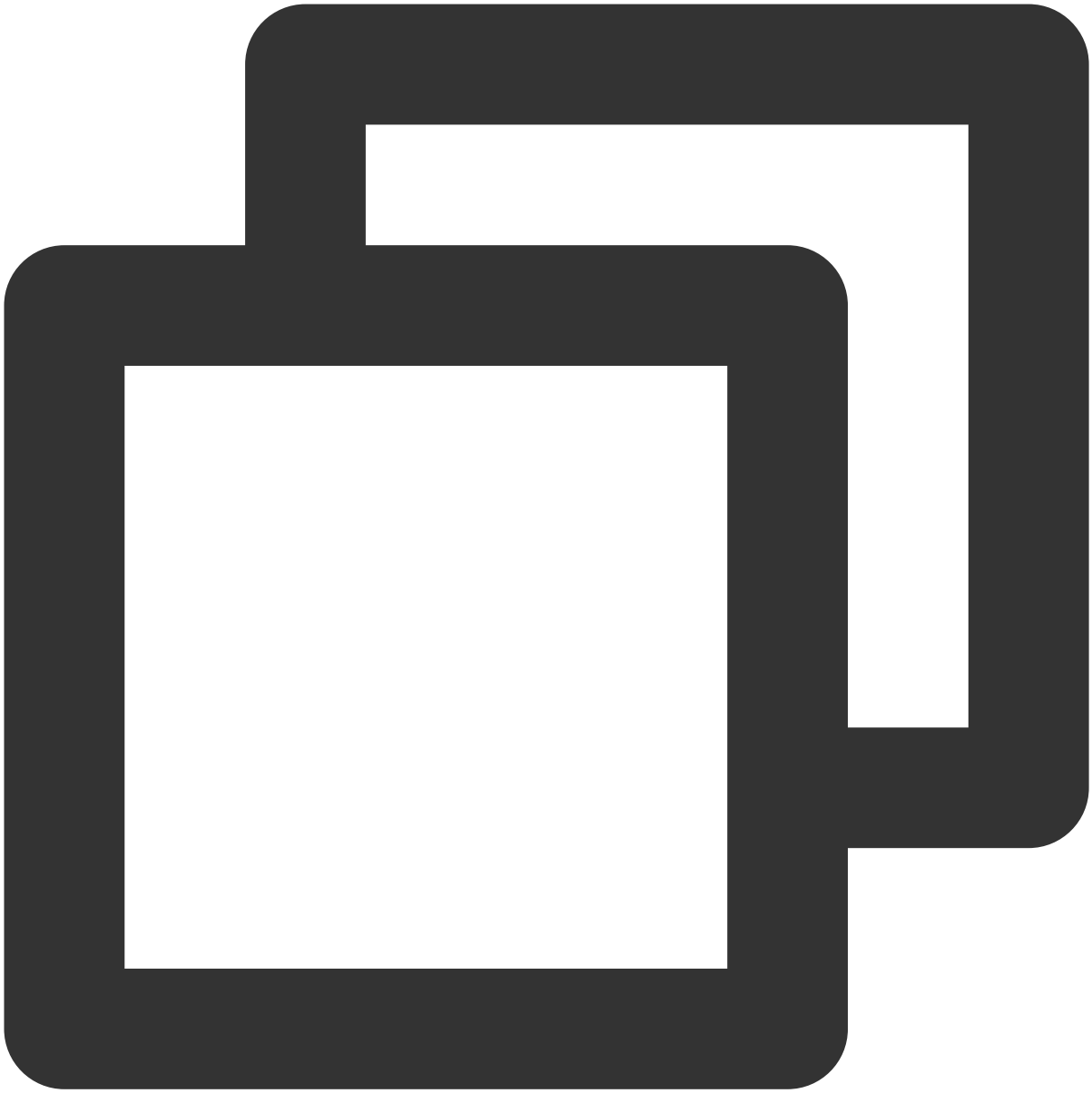
```
"[0-9]{2,}" => The number was 9.9997 but we rounded it off to 10.0.
```



```
"[0-9]{2}" => The number was 9.9997 but we rounded it off to 10.0.
```

字符组

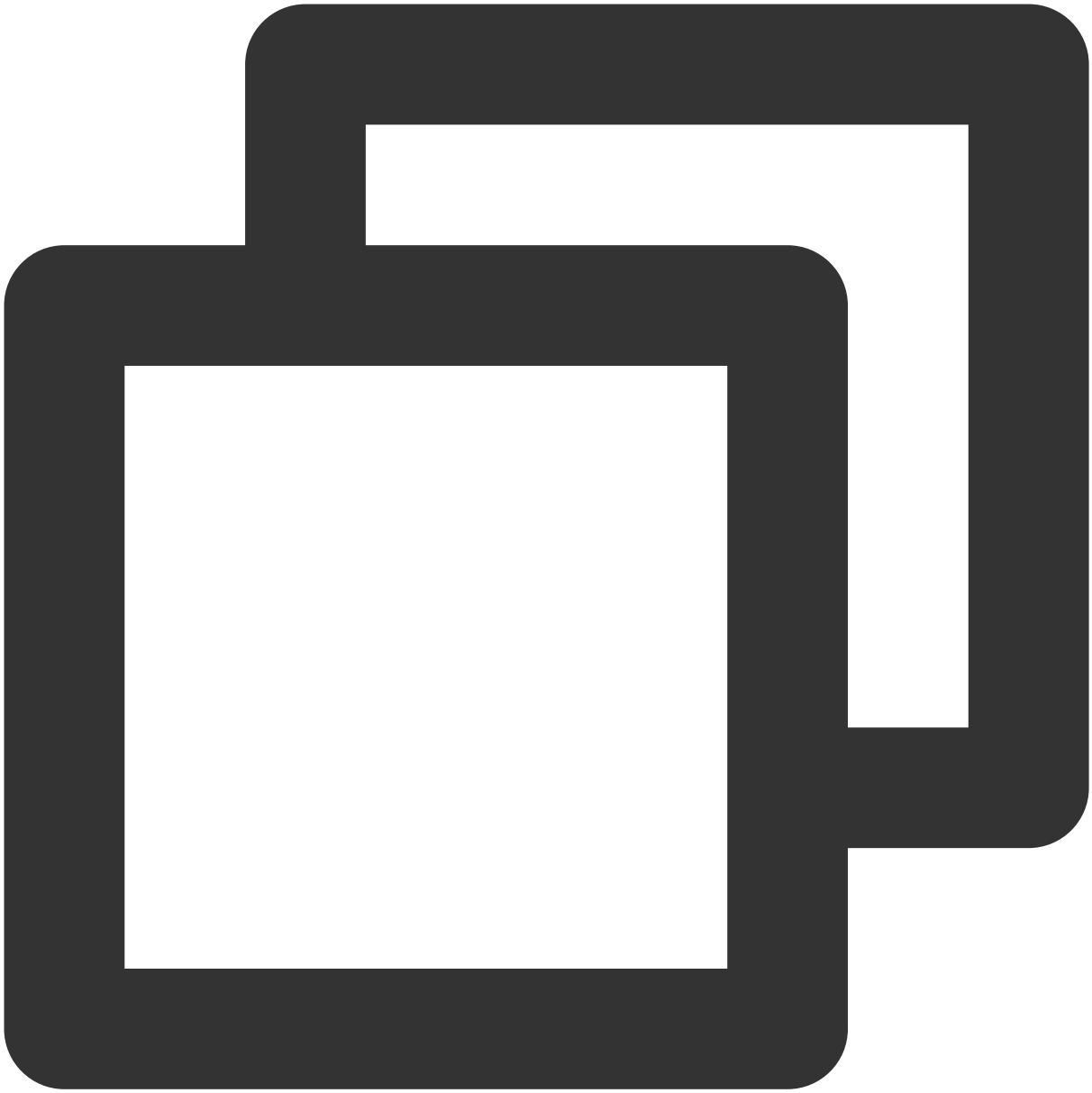
字符组是一组写在圆括号内的子模式 `(...)` 。正如我们在正则表达式中讨论的那样，如果我们把一个量词放在一个字符之后，它会重复前一个字符。
但是，如果我们把量词放在一个字符组之后，它会重复整个字符组。
例如正则表达式 `(ab)*` 表示匹配零个或多个的字符串 "ab"。我们还可以在字符组中使用元字符 `|` 。例如正则表达式 `(c|g|p)ar` ，表示: 小写字母 `c` 、 `g` 或 `p` 后面跟字母 `a` ，后跟字母 `r` 。



```
"(c|g|p)ar" => The car is parked in the garage.
```

分支结构

在正则表达式中垂直条 `|` 用来定义分支结构，分支结构就像多个表达式之间的条件。现在您可能认为这个字符集和分支机构的工作方式一样。
但是字符集和分支结构巨大的区别是字符集只在字符级别上有作用，然而分支结构在表达式级别上依然可以使用。例如正则表达式 `(T|t)he|car` ，表示: 大写字母 `T` 或小写字母 `t` ，后面跟小写字母 `h` ，后跟小写字母 `e` 或小写字母 `c` ，后跟小写字母 `a` ，后跟小写字母 `r` 。

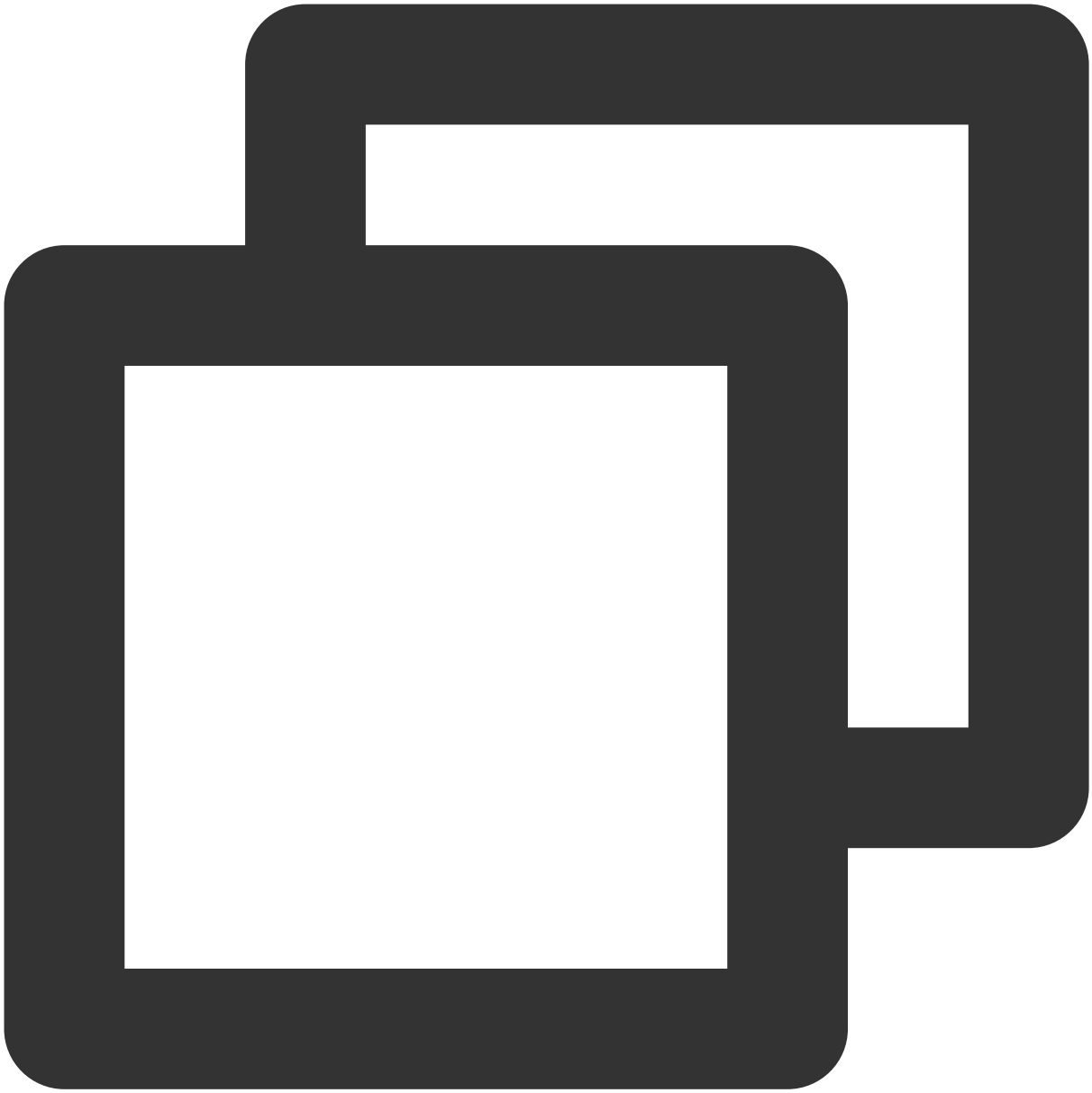


```
"(T|t)he|car" => The car is parked in the garage.
```

转义特殊字符

正则表达式中使用反斜杠 `\\` 来转义下一个字符。这将允许您使用保留字符来作为匹配字符 `{ } [ ] / \\ + * . $ ^ | ?` 。在特殊字符前面加 `\\` ，就可以使用它来做匹配字符。

例如正则表达式 `.` 是用来匹配除了换行符以外的任意字符。现在要在输入字符串中匹配 `.` 字符，正则表达式 `(f|c|m)at\\.?` ，表示: 小写字母 `f` 、 `c` 或者 `m` 后跟小写字母 `a` ，后跟小写字母 `t` ，后跟可选的 `.` 字符。



```
"(f|c|m)at\\.?" => The fat cat sat on the mat.
```

定位符

在正则表达式中，为了检查匹配符号是否是起始符号或结尾符号，我们使用定位符。

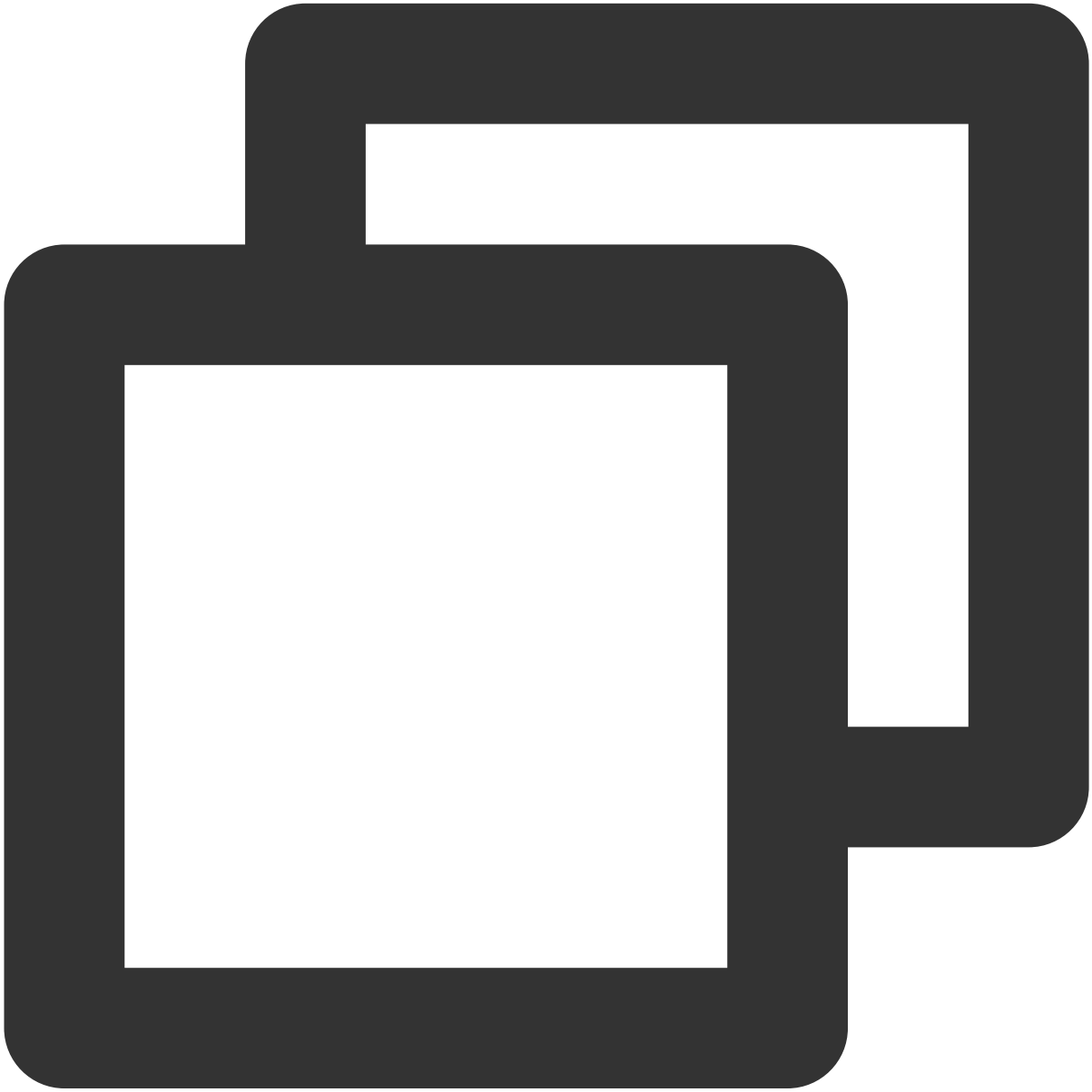
定位符有两种类型: 第一种类型是 `^` 检查匹配字符是否是起始字符，第二种类型是 `$` ，它检查匹配字符是否是输入字符串的最后一个字符。

插入符号

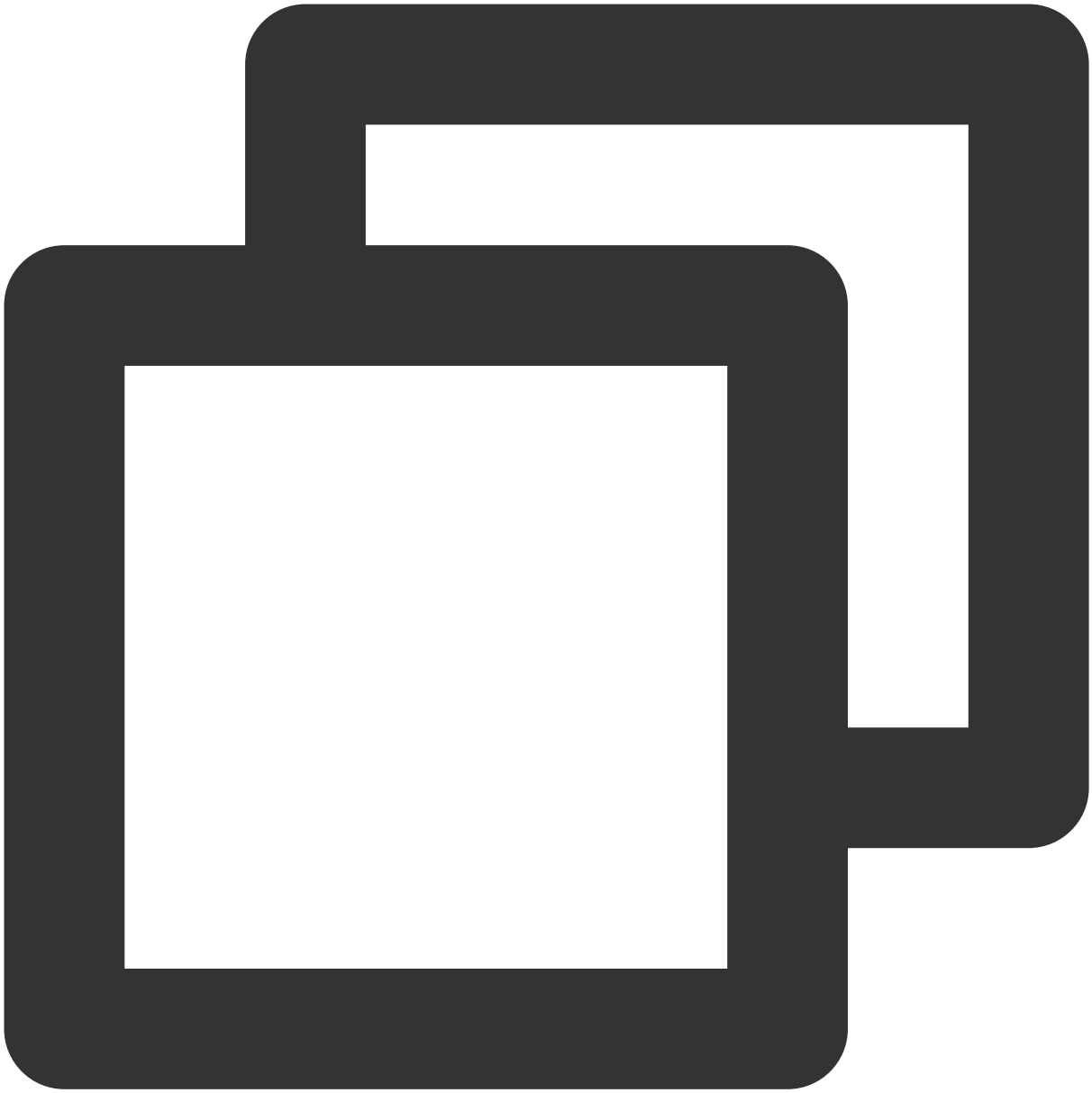
插入符号 `^` 符号用于检查匹配字符是否是输入字符串的第一个字符。如果我们使用正则表达式 `^a` (如果a是起始符号)匹配字符串 `abc`，它会匹配到 `a`。

但是如果我们使用正则表达式 `^b`，它是匹配不到任何东西的，因为在字符串 `abc` 中 "b" 不是起始字符。

让我们来看另一个正则表达式 `^(T|t)he`，这表示: 大写字母 `T` 或小写字母 `t` 是输入字符串的起始符号，后面跟着小写字母 `h`，后跟小写字母 `e`。



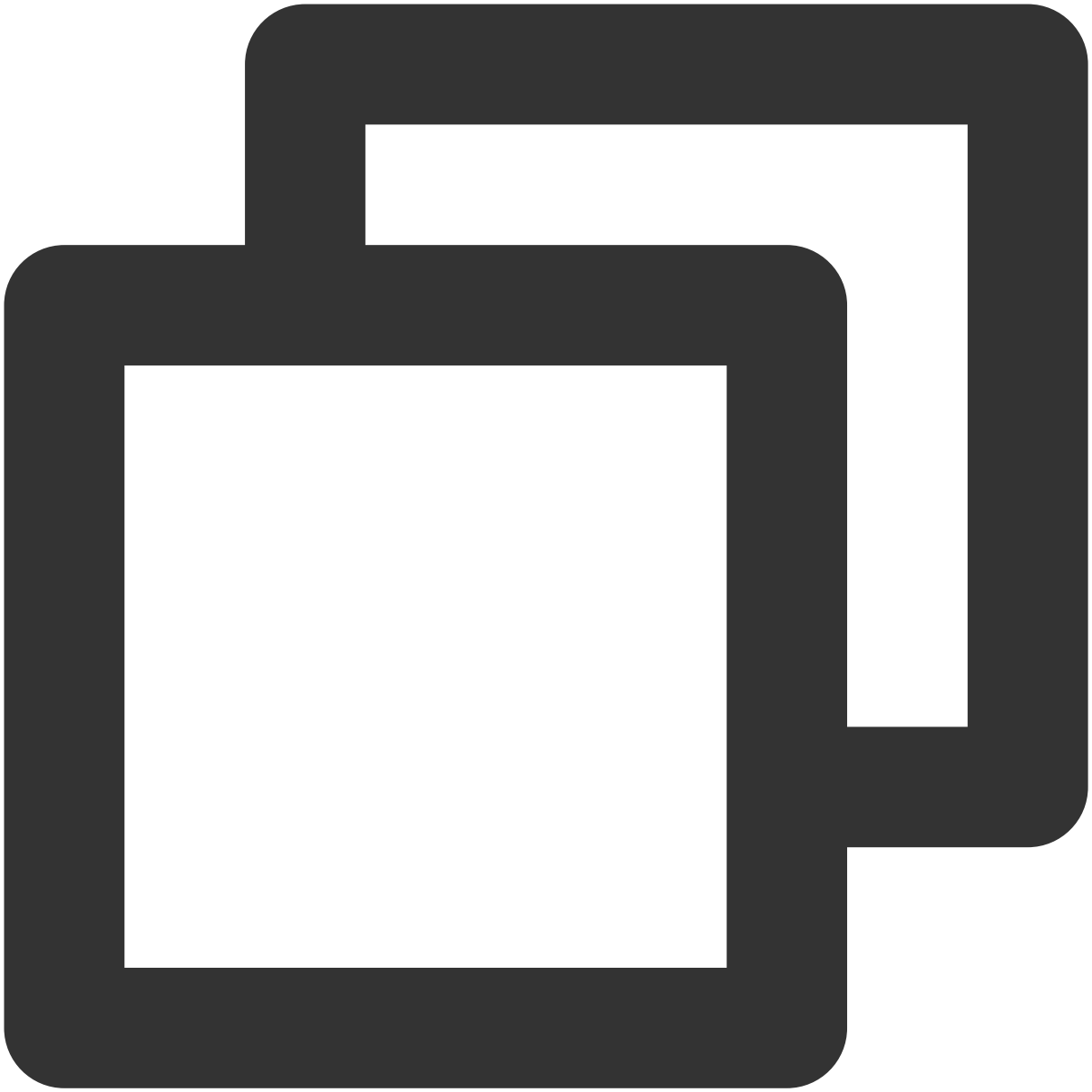
```
"(T|t)he" => The car is parked in the garage.
```



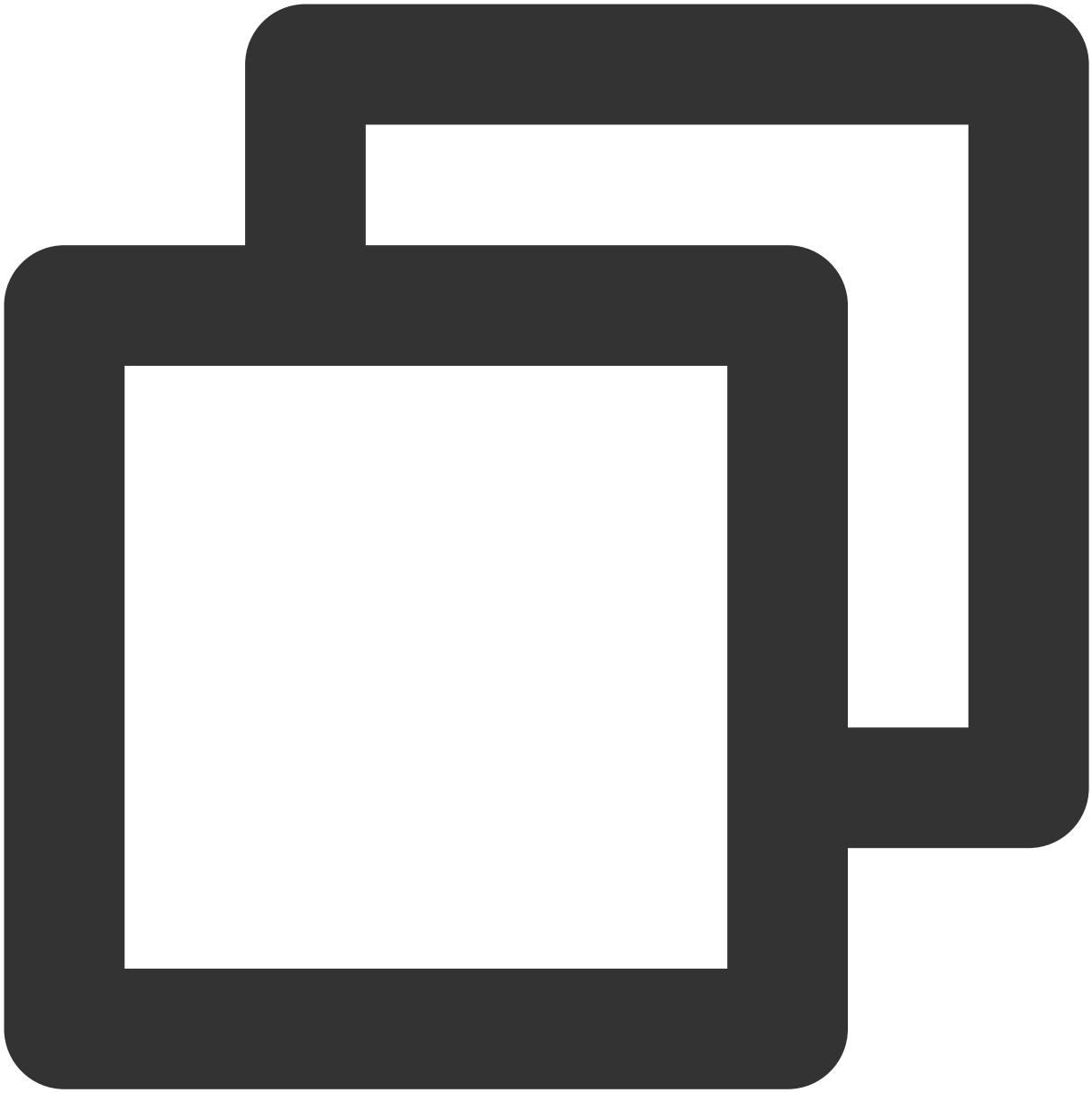
```
"^(T|t)he" => The car is parked in the garage.
```

美元符号

美元 `$` 符号用于检查匹配字符是否是输入字符串的最后一个字符。例如正则表达式 `(at\\.)$`，表示: 小写字母 `a`，后跟小写字母 `t`，后跟一个 `.` 字符，且这个匹配器必须是字符串的结尾。



```
"(at\\.)" => The fat cat. sat. on the mat.
```



```
"(at\\.)$" => The fat cat sat on the mat.
```

简写字符集

正则表达式为常用的字符集和常用的正则表达式提供了简写。简写字符集如下:

简写	描述
.	匹配除换行符以外的任意字符

\\w	匹配所有字母和数字的字符: `[a-zA-Z0-9_]`
\\W	匹配非字母和数字的字符: `[^\\w]`
\\d	匹配数字: `[0-9]`
\\D	匹配非数字: `[^\\d]`
\\s	匹配空格符: `[\\t\\n\\f\\r\\p{Z}]`
\\S	匹配非空格符: `[^\\s]`

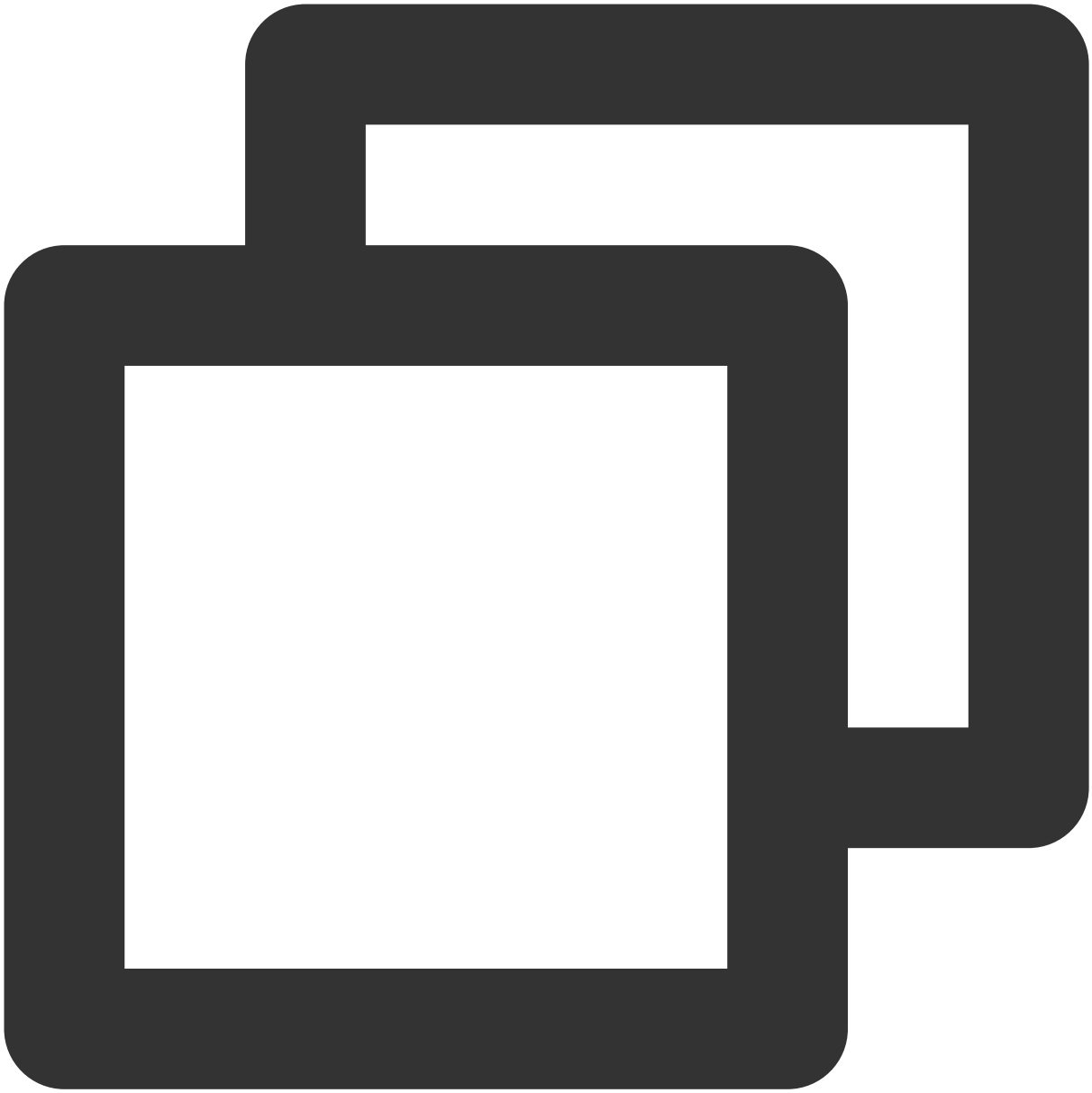
断言

后行断言和先行断言有时候被称为断言，它们是特殊类型的 ***非捕获组*** (用于匹配模式，但不包括在匹配列表中)。当我们在一种特定模式之前或者之后有这种模式时，会优先使用断言。
例如我们想获取输入字符串 `$4.44 and $10.88` 中 `$` 字符之前的所有数字。我们可以使用这个正则表达式 `(?<=\\$)[0-9\\.]*` ，表示: 获取 `$` 字符之前的所有的数字包含 `.` 字符。
以下是正则表达式中使用的断言:

符号	描述
?=	正向先行断言
?!	负向先行断言
?<=	正向后行断言
?<!	负向后行断言

正向先行断言

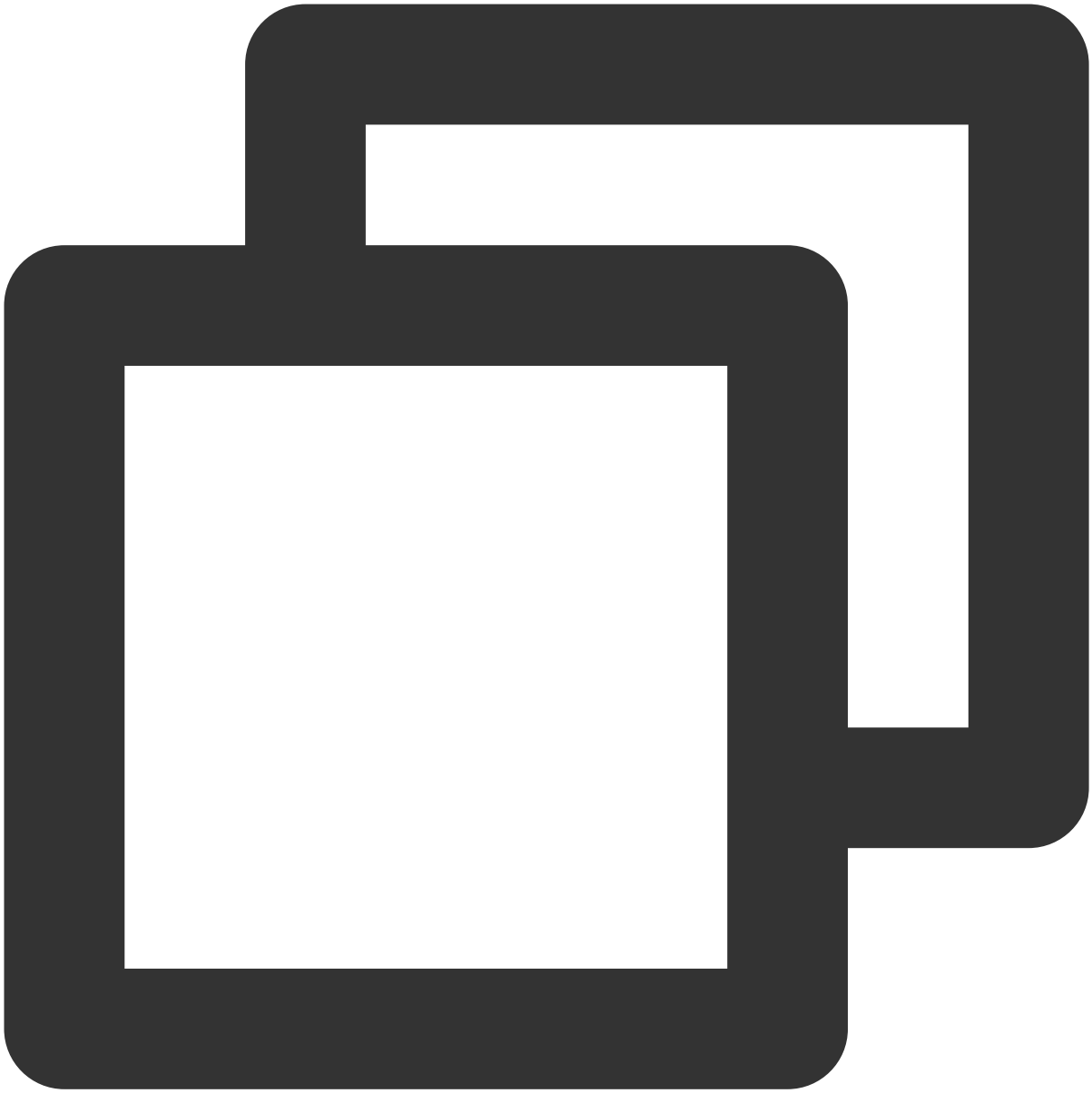
正向先行断言认为第一部分的表达式必须是先行断言表达式。返回的匹配结果仅包含与第一部分表达式匹配的文本。
要在一个括号内定义一个正向先行断言，在括号中问号和等号是这样使用的 `(?=...)` 。先行断言表达式写在括号中的等号后面。
例如正则表达式 `(T|t)he(?=\\sfat)` ，表示: 匹配大写字母 `T` 或小写字母 `t` ，后面跟字母 `h` ，后跟字母 `e` 。
在括号中，我们定义了正向先行断言，它会引导正则表达式引擎匹配 `The` 或 `the` 后面跟着 `fat` 。



```
"(at\\.)$" => The fat cat sat on the mat.
```

负向先行断言

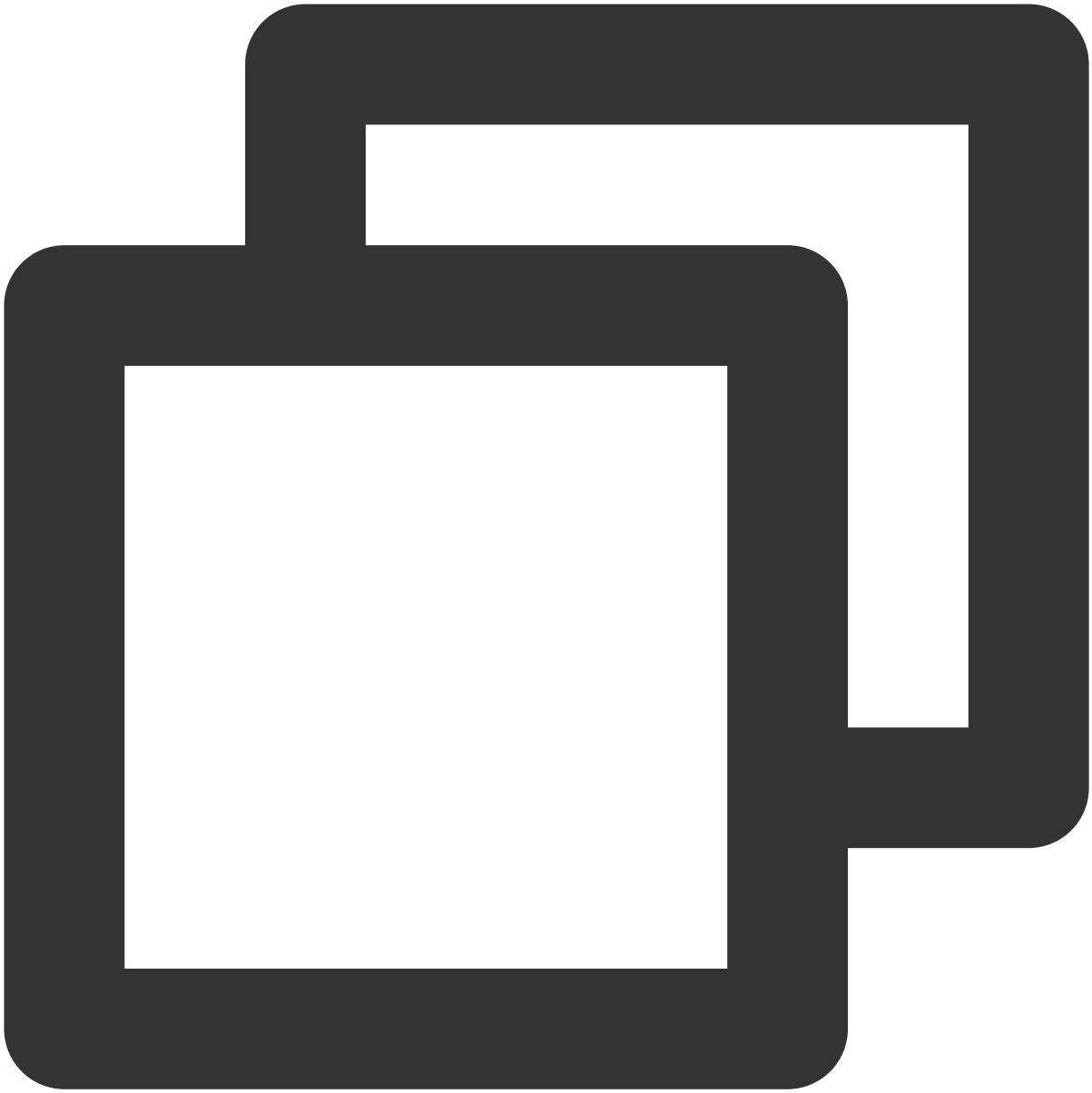
当我们需要从输入字符串中获取不匹配表达式的内容时，使用负向先行断言。负向先行断言的定义跟我们定义的正向先行断言一样，
唯一的区别是不是等号 `=`，我们使用否定符号 `!`，例如 `(?!...)`。
我们来看下面的正则表达式 `(T|t)he(?!\\sfat)`，表示: 从输入字符串中获取全部 `The` 或者 `the` 且不匹配 `fat` 前面加上一个空格字符。



```
"(T|t)he(?!\\sfat)" => The fat cat sat on the mat.
```

正向后行断言

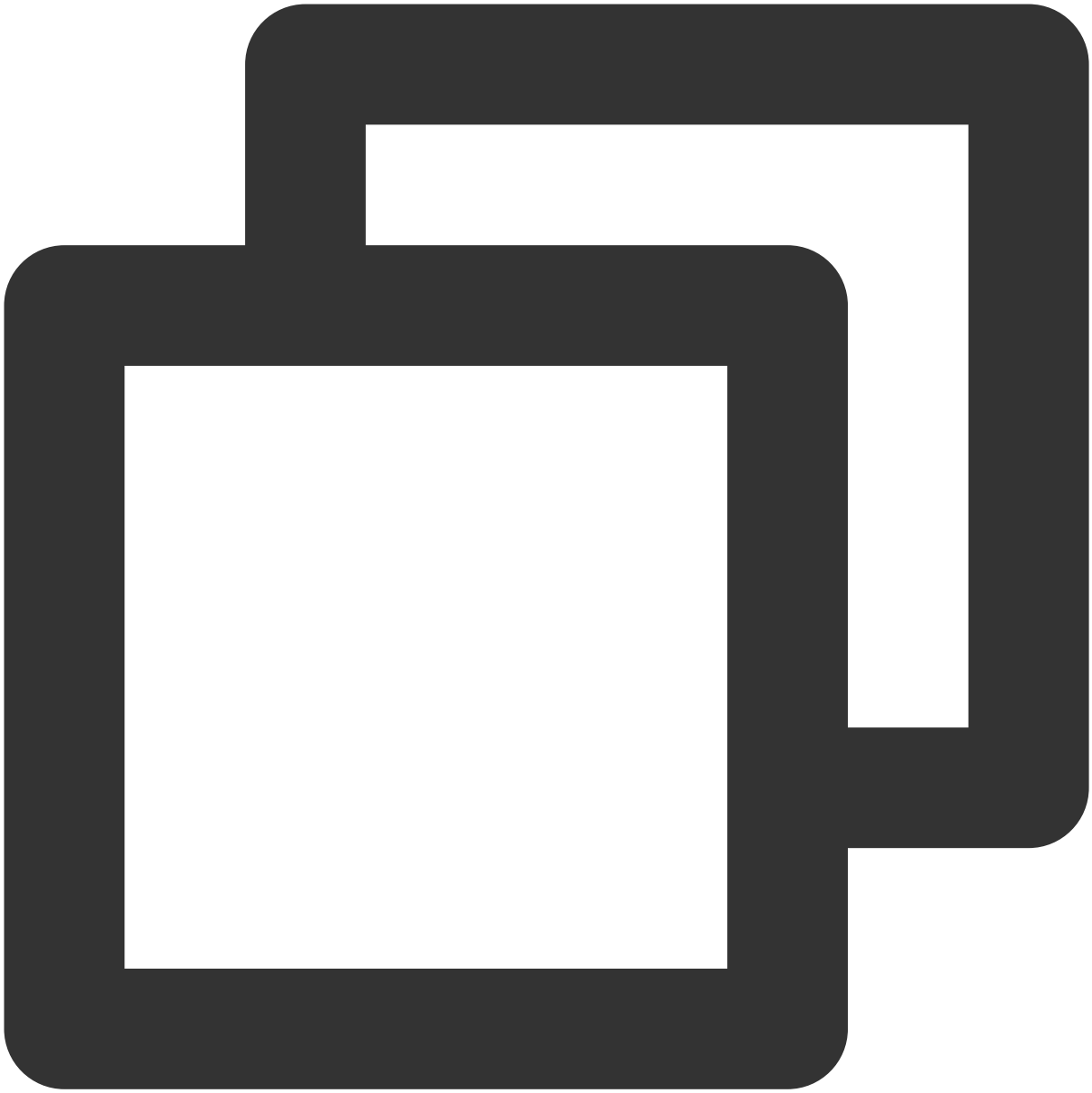
正向后行断言是用于获取在特定模式之前的所有匹配内容。正向后行断言表示为 `(?<=...)` 。例如正则表达式 `(?<=(T|t)he\\s)(fat|mat)` ，表示: 从输入字符串中获取在单词 `The` 或 `the` 之后的所有 `fat` 和 `mat` 单词。



```
"(?<=(T|t)he\\s)(fat|mat)" => The fat cat sat on the mat.
```

负向后行断言

负向后行断言是用于获取不在特定模式之前的所有匹配的内容。负向后行断言表示为 `(?<!...)` 。例如正则表达式 `(?<!(T|t)he\\s)(cat)` ，表示: 在输入字符中获取所有不在 `The` 或 `the` 之后的所有单词 `cat` 。



```
"(?<!(T|t)he\\s)(cat)" => The cat sat on cat.
```

标记

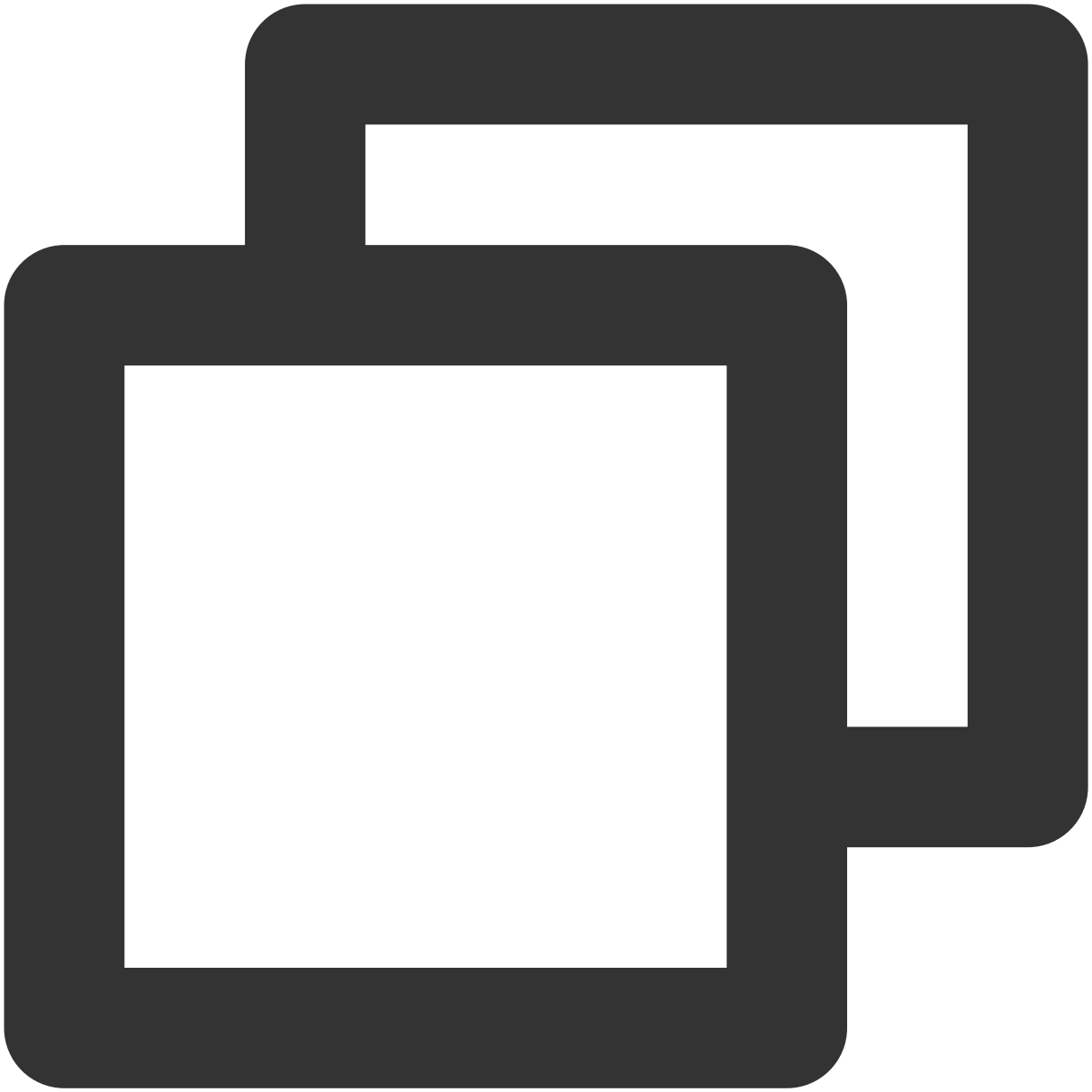
标记也称为修饰符，因为它会修改正则表达式的输出。这些标志可以以任意顺序或组合使用，并且是正则表达式的一部分。

标记	描述

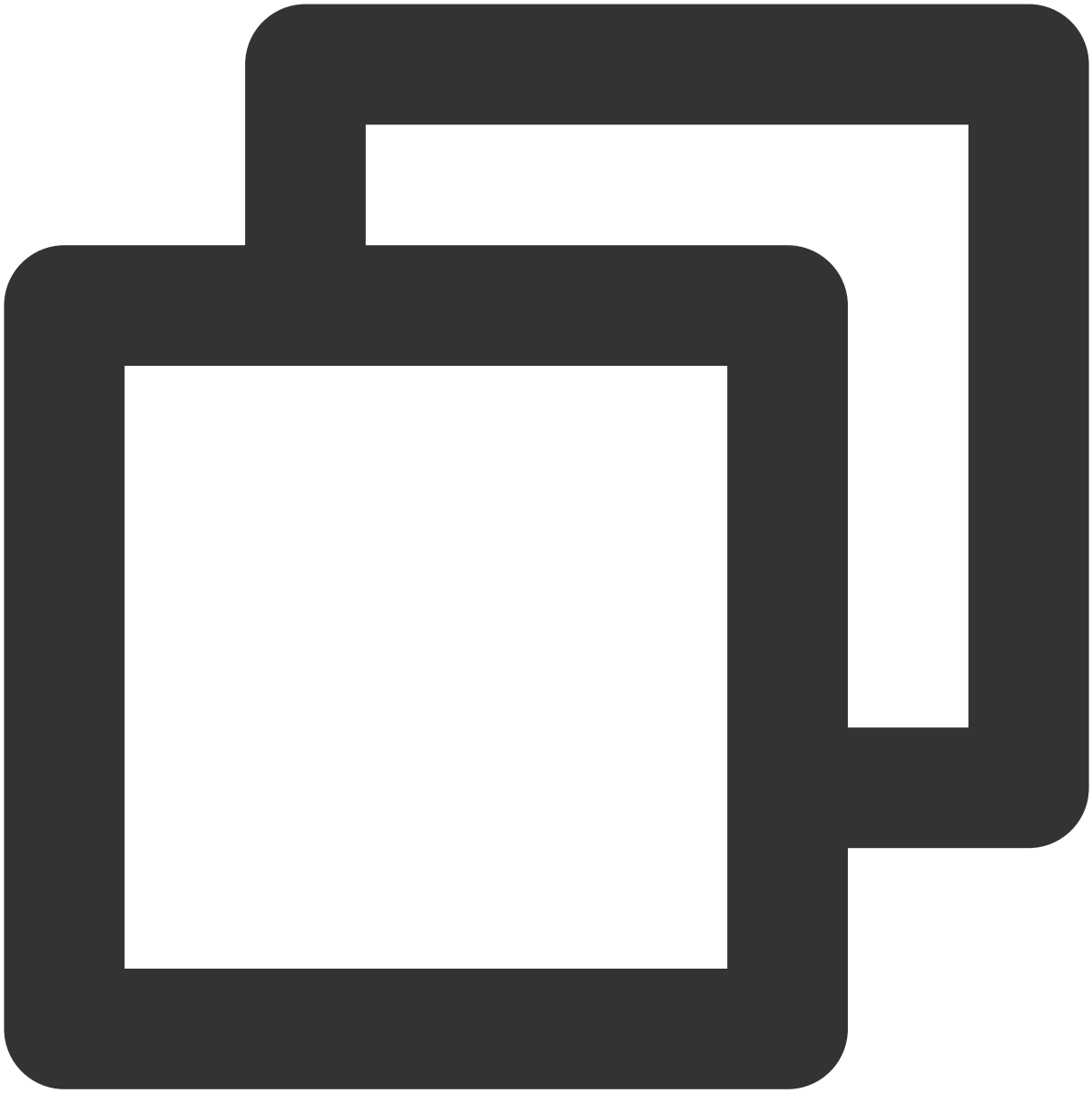
i	不区分大小写: 将匹配设置为不区分大小写。
g	全局搜索: 搜索整个输入字符串中的所有匹配。
m	多行匹配: 会匹配输入字符串每一行。

不区分大小写

`i` 修饰符用于执行不区分大小写匹配。例如正则表达式 `/The/gi`，表示: 大写字母 `T`，后跟小写字母 `h`，后跟字母 `e`。
但是在正则匹配结束时 `i` 标记会告诉正则表达式引擎忽略这种情况。正如您所看到的，我们还使用了 `g` 标记，因为我们要在整个输入字符串中搜索匹配。



```
"The" => The fat cat sat on the mat.
```



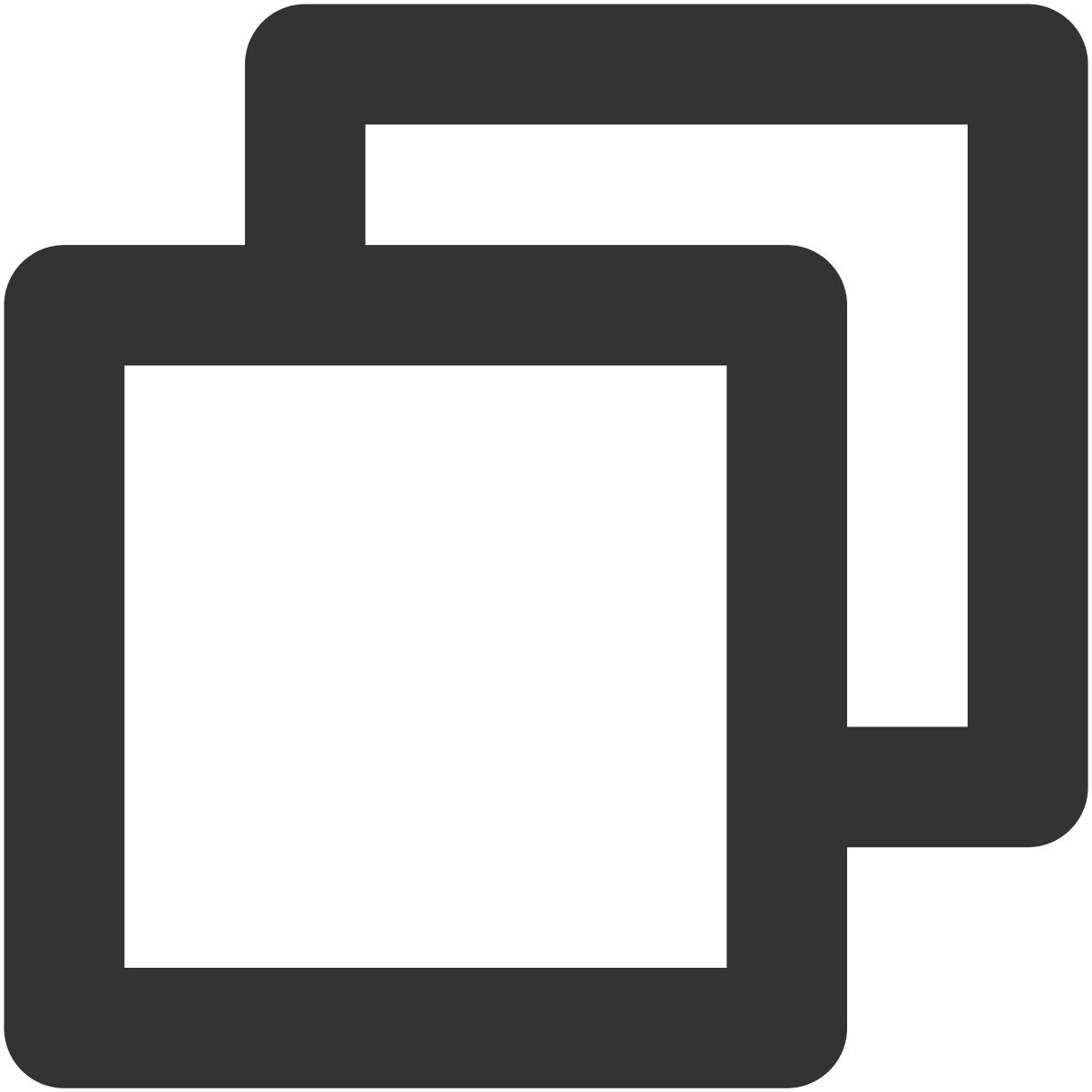
```
"/The/gi" => The fat cat sat on the mat.
```

全局搜索

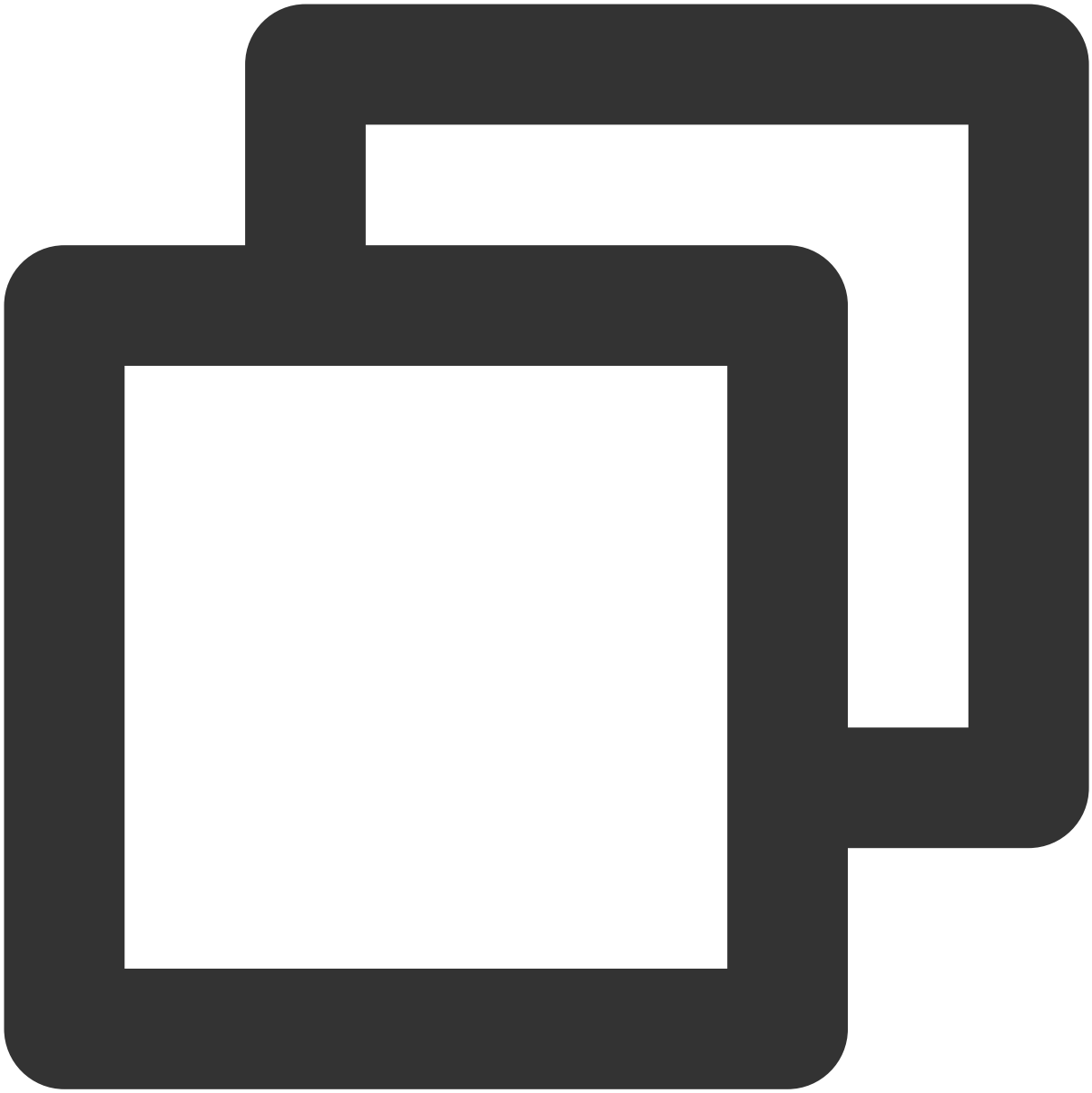
`g` 修饰符用于执行全局匹配 (会查找所有匹配，不会在查找到第一个匹配时就停止)。

例如正则表达式 `/.(at)/g` ，表示: 除换行符之外的任意字符，后跟小写字母 `a` ，后跟小写字母 `t` 。

因为我们在正则表达式的末尾使用了 `g` 标记，它会从整个输入字符串中找到每个匹配项。



```
".(at)" => The fat cat sat on the mat.
```

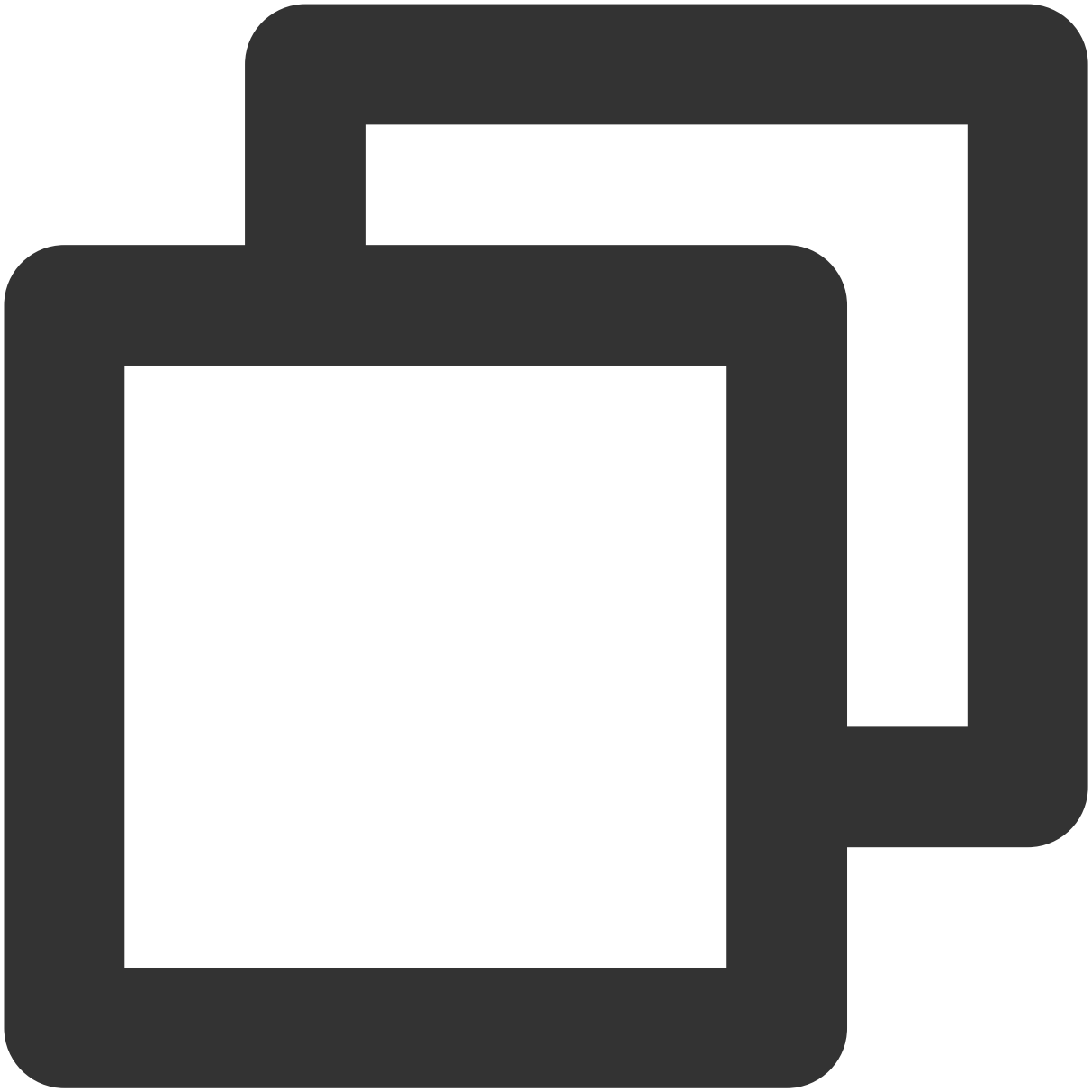


```
"/.(at)/g" => The fat cat sat on the mat.
```

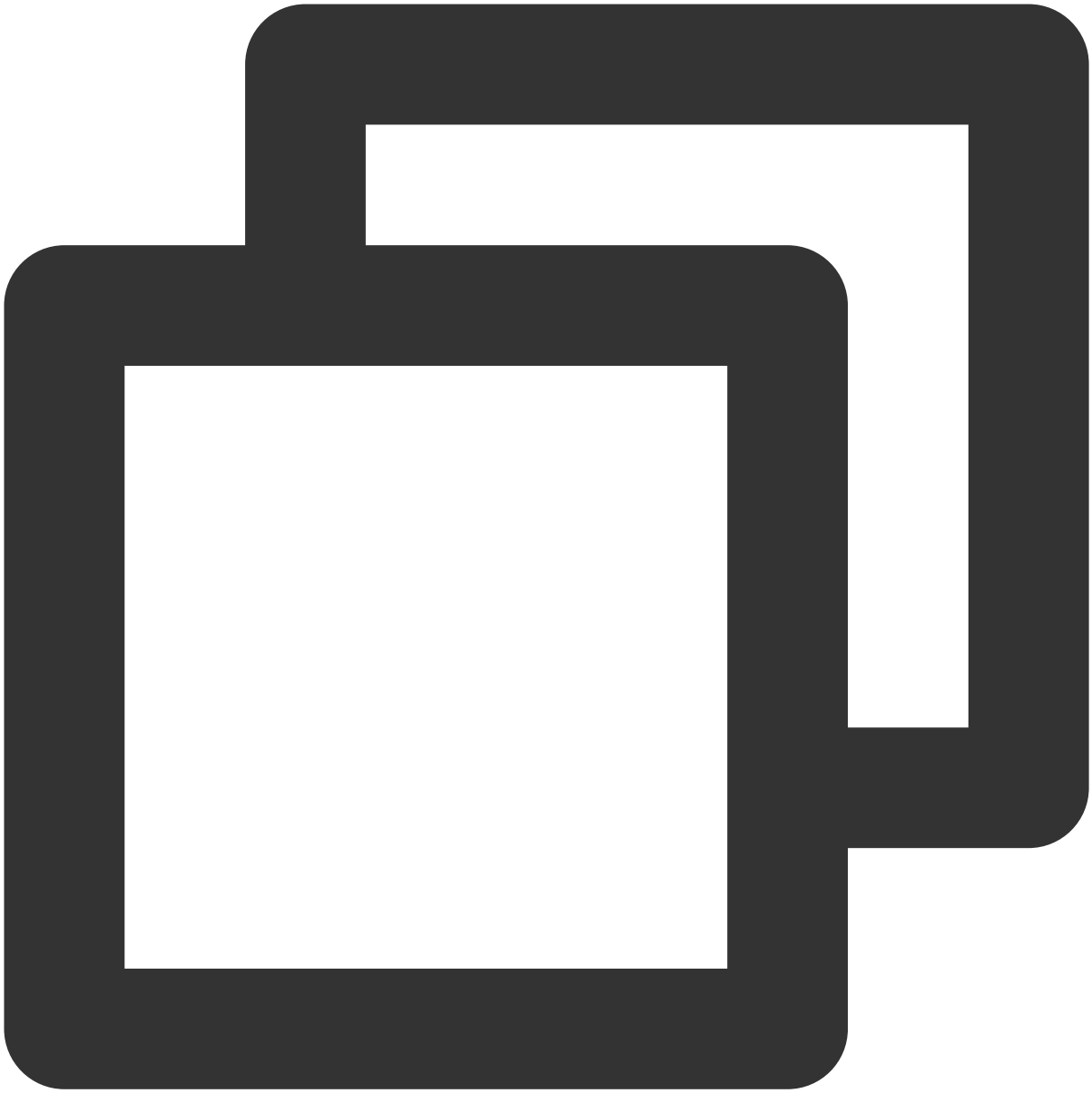
多行匹配

`m` 修饰符被用来执行多行的匹配。正如我们前面讨论过的 `(^, $)`，使用定位符来检查匹配字符是输入字符串开始或者结束。但是我们希望每一行都使用定位符，所以我们就使用 `m` 修饰符。

例如正则表达式 `/at(.)?$/gm`，表示: 小写字母 `a`，后跟小写字母 `t`，匹配除了换行符以外任意字符零次或一次。而且因为 `m` 标记，现在正则表达式引擎匹配字符串中每一行的末尾。



```
"/.at(.)?$/" => The fat
              cat sat

        on the mat.
```



```
"/.at(.)?$/gm" => The fat
                  cat sat
                  on the mat.
```

常用正则表达式

类型	表达式

正整数	^-\\d+$
负整数	^-\\d+$
电话号码	^+?[\\d\\s]{3,}$
电话代码	^+?[\\d\\s]+(?[\\d\\s]{10,}$
整数	^-?\\d+$
用户名	^[\\w\\d_.]{4,16}$
字母数字字符	^[a-zA-Z0-9]*$
带空格的字母数字字符	^[a-zA-Z0-9]*$
密码	^(?=^.{6,}$)((?=.*[A-Za-z0-9])(?=.*[A-Z])(?=.*[a-z]))^.*$
电子邮件	^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4})*$
IPv4 地址	^((?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))*$`
小写字母	^([a-z])*$
大写字母	^([A-Z])*$
用户名	^[\\w\\d_.]{4,16}$
网址	^(((http|https|ftp):\\/\\/)?([[a-zA-Z0-9]\\-\\.])+(\\.)([[a-zA-Z0-9]]){2,4}([[a-zA-Z0-9]\\/+=%&_\\.~?\\-]*))*$
VISA 信用卡号码	^(4[0-9]{12}(?:[0-9]{3})?)*$
日期 (MM/DD/YYYY)	^(0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01])[- /.](19|20)?[0-9]{2}$
日期 (YYYY/MM/DD)	^(19|20)?[0-9]{2}[- /.](0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01])$
万事达信用卡号码	^(5[1-5][0-9]{14})*$