请求体 - 更新数据

用 `PUT` 更新数据

更新数据请用 HTTP `PUT` 操作。

把输入数据转换为以 JSON 格式存储的数据（比如，使用 NoSQL 数据库时），可以使用 `jsonable_encoder`。例如，把 `datetime` 转换为 `str`。

```
{!../../../docs_src/body_updates/tutorial001.py!}
```

`PUT` 用于接收替换现有数据的数据。

关于更新数据的警告

用 `PUT` 把数据项 `bar` 更新为以下内容时：

```
{
    "name": "Barz",
    "price": 3,
    "description": None,
}
```

因为上述数据未包含已存储的属性 `"tax": 20.2`，新的输入模型会把 `"tax": 10.5` 作为默认值。

因此，本次操作把 `tax` 的值「更新」为 `10.5`。

用 `PATCH` 进行部分更新

HTTP `PATCH` 操作用于更新 *部分* 数据。

即，只发送要更新的数据，其余数据保持不变。

!!! Note "笔记"

`PATCH` 没有 `PUT` 知名，也怎么不常用。

很多人甚至只用 `PUT` 实现部分更新。

FastAPI 对此没有任何限制，可以**随意**互换使用这两种操作。

但本指南也会分别介绍这两种操作各自的用途。

使用 Pydantic 的 `exclude_unset` 参数

更新部分数据时，可以在 Pydantic 模型的 `.dict()` 中使用 `exclude_unset` 参数。

比如，`item.dict(exclude_unset=True)`。

这段代码生成的 `dict` 只包含创建 `item` 模型时显式设置的数据，而不包括默认值。

然后再用它生成一个只含已设置（在请求中所发送）数据，且省略了默认值的 `dict`：

```
{!../../../docs_src/body_updates/tutorial002.py!}
```

使用 Pydantic 的 `update` 参数

接下来，用 `.copy()` 为已有模型创建调用 `update` 参数的副本，该参数为包含更新数据的 `dict`。

例如，`stored_item_model.copy(update=update_data)`：

```
{!../../../docs_src/body_updates/tutorial002.py!}
```

更新部分数据小结

简而言之，更新部分数据应：

- 使用 `PATCH` 而不是 `PUT`（可选，也可以用 `PUT`）；
- 提取存储的数据；
- 把数据放入 Pydantic 模型；
- 生成不含输入模型默认值的 `dict`（使用 `exclude_unset` 参数）；
    - 只更新用户设置过的值，不用模型中的默认值覆盖已存储过的值。
- 为已存储的模型创建副本，用接收的数据更新其属性（使用 `update` 参数）。
- 把模型副本转换为可存入数据库的形式（比如，使用 `jsonable_encoder`）。
    - 这种方式与 Pydantic 模型的 `.dict()` 方法类似，但能确保把值转换为适配 JSON 的数据类型，例如，把 `datetime` 转换为 `str`。
- 把数据保存至数据库；
- 返回更新后的模型。

```
{!../../../docs_src/body_updates/tutorial002.py!}
```

!!! tip "提示"

实际上，`HTTP `PUT`` 也可以完成相同的操作。
但本节以 `` `PATCH` `` 为例的原因是，该操作就是为了这种用例创建的。

!!! note "笔记"

注意，输入模型仍需验证。

因此，如果希望接收的部分更新数据可以省略其他所有属性，则要把模型中所有的属性标记为可选（使用默认值或 `` `None` ``）。

为了区分用于**更新**所有可选值的模型与用于**创建**包含必选值的模型，请参照[更多模型](extra-models.md){.internal-link target=_blank} 一节中的思路。