

TypeScript Template Literals

Template literals understandable

Motivation

Template literal types build on **string literal types**, and have the ability to **expand into many strings** via unions.

Template Literal Types

We have already seen template literal types during the basics chapter.

Since they have some quite interesting aspects, let us talk about them a little more.

Template Literal Types

Following a simple example of a template literal type:

```
1 // 07_template-literals/listings/00_greeting.ts
2
3 type World = "world";
4 type Greeting = `hello ${World}`;
5 // ^ = type Greeting = "hello world"
```

Please keep in mind, this is only a type.

Unions

The former example does not add much use and is better done with **enums**. But sometimes you have a whole dictionary of words, **too complicated for enums**.

Here you can use **Template Literal Types together with unions**:

```
1 // 07_template-literals/listings/01_unions.ts
2
3 type EmailLocaleIDs = "welcome_email" | "email_heading";
4 type FooterLocaleIDs = "footer_title" | "footer_sendoff";
5 type AllLocaleIDs = `${EmailLocaleIDs | FooterLocaleIDs}_id`;
```

Unions

```
1 // 07_template-literas/listings/01_unions.ts
2
3 type EmailLocaleIDs = "welcome_email" | "email_heading";
4 type FooterLocaleIDs = "footer_title" | "footer_sendoff";
5 type AllLocaleIDs = `${EmailLocaleIDs | FooterLocaleIDs}_id`;
```

Question: What would be acceptable for **AllLocaleIDs**?

Unions

```
1 // 07_template-literals/listings/01_unions.ts
2
3 type EmailLocaleIDs = "welcome_email" | "email_heading";
4 type FooterLocaleIDs = "footer_title" | "footer_sendoff";
5 type AllLocaleIDs = `${EmailLocaleIDs | FooterLocaleIDs}_id`;
```

Question: What would be acceptable for **AllLocaleIDs**?

Solution: **welcome_email_id** or **email_heading_id** or **footer_title_id** or **footer_sendoff_id**.

Unions

For each interpolated position in the template literal, the unions are cross multiplied:

```
1 // 07_template-literals/listings/02_unions_crossproduct.ts
2
3 type AllLocaleIDs = `${EmailLocaleIDs | FooterLocaleIDs}_id`;
4 type Lang = "en" | "de" | "fr";
5 type LocaleMessageIDs = `${Lang}_${AllLocaleIDs}`;
```

We will get `en_welcome_email_id`, `en_email_heading`, `en_footer_title` and `en_footer_sendoff` for each lang.

String Union in Types

The power in template literals comes when defining a new string based off an existing string inside a type.

Very common:

- extend an object based on the fields that it currently has.

String Union in Types

Please have a look at the following example:

```
1 // 07_template-literas/listings/03_unions_in_types.ts
2
3 type PropEventSource<T> = {
4   on(eventName: `${string & keyof T}Changed`, callback: () => void): void;
5 };
6 declare function makeWatchedObject<T>(obj: T): T & PropEventSource<T>;
7
8 let person = makeWatchedObject({firstName: "David",});
9
10 // error!
11 person.on("firstName", () => {
12 });
13 // error!
14 person.on("firstNameChanged", () => {});
15 // success!
16 person.on("firstNameChanged", () => {console.log(`firstName was changed!`)});
17 });
```

String Union in Types

Let us have a deeper insight into what happens:

```
1 // 07_template-literas/listings/03_unions_in_types.ts
2
3 type PropEventSource<T> = {
4   on(eventName: `${string & keyof T}Changed`, callback: () => void): void;
5 };
```

We first create a type that enforces the later change to be named ...Changed and being a void callback.

String Union in Types

Then we create a "watched object" with an 'on' method, so that we can watch for changes to properties.



```
1 // 07_template-literas/listings/03_unions_in_types.ts
2
3 declare function makeWatchedObject<T>(obj: T): T & PropEventSource<T>;
```

String Union in Types

After that we need an object (person), and some .on methods

```
1 // 07_template-literas/listings/03_unions_in_types.ts
2
3 let person = makeWatchedObject({
4     firstName: "David",
5 });
6
7 // error!
8 person.on("firstName", () => {});
9
10 // error!
11 person.on("fristNameChanged", () => {});
12
13 // success!
14 person.on("firstNameChanged", () => {console.log(`firstName was changed!`)});
15 });
```

The only one which fits with our expectation is the last one.

Great Job!

That was all for template literals!



End

That was all for this chapter
