Angular 21

THE COMPLETE GUIDE

Component Communication

Re-imagined with Signals

`input()` `output()` `model()`

@cb0401

Read on →

Evolution

Legacy vs. Modern

Legacy @Input

```
@Input() count = 0;

ngOnChanges() {
  // Manual reaction
  this.double = this.count
}
```

Signal input()

```
count = input(0);

// Auto reaction
double = computed(() =>
  this.count() * 2
);
```

The Basic Syntax

The `input()` function returns a read-only Signal. It's concise and type-safe.

```
// Simple Input
name = input<string>();

// Input with type inference
age = input<number>();
```

Because it's a signal, you read it as `this.age()`. No more accidental mutations inside the component!

Safety

Strict Contracts

Ensure parent components **must** provide data. If they don't, the app won't pile.

```
// Required Input
email = input.required<string>();


// Usage in Parent Template
<app-signal-user-input [email]="'john@test.com'" />
```

- ✓ No more `undefined` checks
- ✓ No more `!` non-null assertions
- ✓ Self-documenting API

Input Transforms

Parse data at the boundary. Perfect for boolean attributes.

```
// Input with transform
displayName = input('', {
  transform: (v: string) => v.toUpperCase(),
});
```

```
<app-comp [displayName]="'john'" />   // Becomes "JOHN"
```

Modern Outputs

Goodbye `@Output` + `EventEmitter`. Hello `output()`.

```
export class CounterOutput {
  counter = signal(0);

  //    Basic output
  countChanged = output<number>();

  increment() {
    this.counter.update(c => c + 1);
    this.countChanged.emit(this.counter());
  }
}
```

It creates an instance of `OutputEmitterRef`. It's not a Signal itself (outputs are events, not state), but it fits the new style.

Advanced

Complex Payloads

Outputs support void for triggers and complex objects for data.

```
//    Output for events without data
resetClicked = output<void>();

//    Output with complex data
actionPerformed = output<{
  action: 'increment' | 'reset';
  value: number;
  timestamp: Date;
}>();
```

Two-Way

The Power of Model

The `model()` primitive is special. It's a writable signal that communicates back to the parent.

```
// Child Component
checked = model(false);

toggle() {
  // Updates local state AND notify parent
  this.checked.update(c => !c);
}
```

```
<app-toggle [(checked)]="isValid" />
```

@cb0401

Aliasing Props

Keep your internal code clean while supporting legacy public APIs.

```
// Default Input with alias
role = input<string>('USER', {
  alias: 'userRole'
});

// Output with alias
valueChanged = output<number>({
  alias: 'onValueChange'
});
```

Useful when refactoring large applications without breaking template contracts.

Summary

Quick Cheat Sheet

input()
Read-only from Parent

output()
Event to Parent

model()
Read/Write (Two-Way)

Ready to Refactor?

Signals are not just a new feature; they are a new paradigm for reactivity.

Next Module: **Dependancy Injection**

Found this helpful?
Save it for later!