Asynchronously Reading Resources

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

Adding an asynchronous controller action

Testing the improvements from writing async code

Using result filters for shaping action results

Demo

Getting resources

Introducing WebSurge

Our goal is to test for scalability improvements

- Load testing
- Combined with thread pool throttling

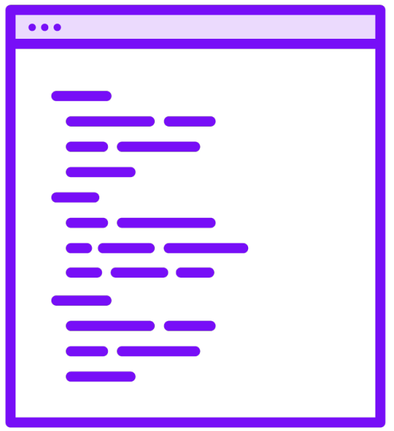
WebSurge is a tool specifically aimed at load testing

Demo

Using WebSurge to test async code improvements

The Outer Facing Model

Entity model
Entity classes represent (partial) database rows as objects



Outer facing model
DTO classes represent resources that are sent over the wire

The Outer Facing Model

Book entity

```
Guid Id
string Title
string Description
Guid AuthorId
Author Author
```

Book DTO

```
Guid Id
string Title
string Description
```

The Outer Facing Model

Book entity

```
Guid Id
string Title
string Description
Guid AuthorId
Author Author
```

Book DTO

```
Guid Id
string Title
string Description
```

The Outer Facing Model

Book entity

```
Guid Id
string Title
string Description
Guid AuthorId
Author Author
```

Book DTO

```
Guid Id
string Title
string Description
```

The Outer Facing Model

Book entity

```
Guid Id
string Title
string Description
Guid AuthorId
Author Author
```

Book DTO

```
Guid Id
string Title
string Description
string AuthorName
```

The Outer Facing Model

Book entity

```
Guid Id
string Title
string Description
Guid AuthorId
Author Author
```

Book DTO

```
Guid Id
string Title
string Description
string AuthorName
IEnumerable<BookCover>
BookCovers
```

Mixing models & responsibilities between layers leads to evolvability issues

The Outer Facing Model

How do we represent the resource data type?

Model classes (DTOs)
Statically typed approach

Dynamics, anonymous objects, ExpandoObject
Dynamically typed approach

Mapping code in controller actions

The Outer Facing Model

How do we represent the resource data type?

Model classes (DTOs)
Statically typed approach

Dynamics, anonymous objects, ExpandoObject
Dynamically typed approach

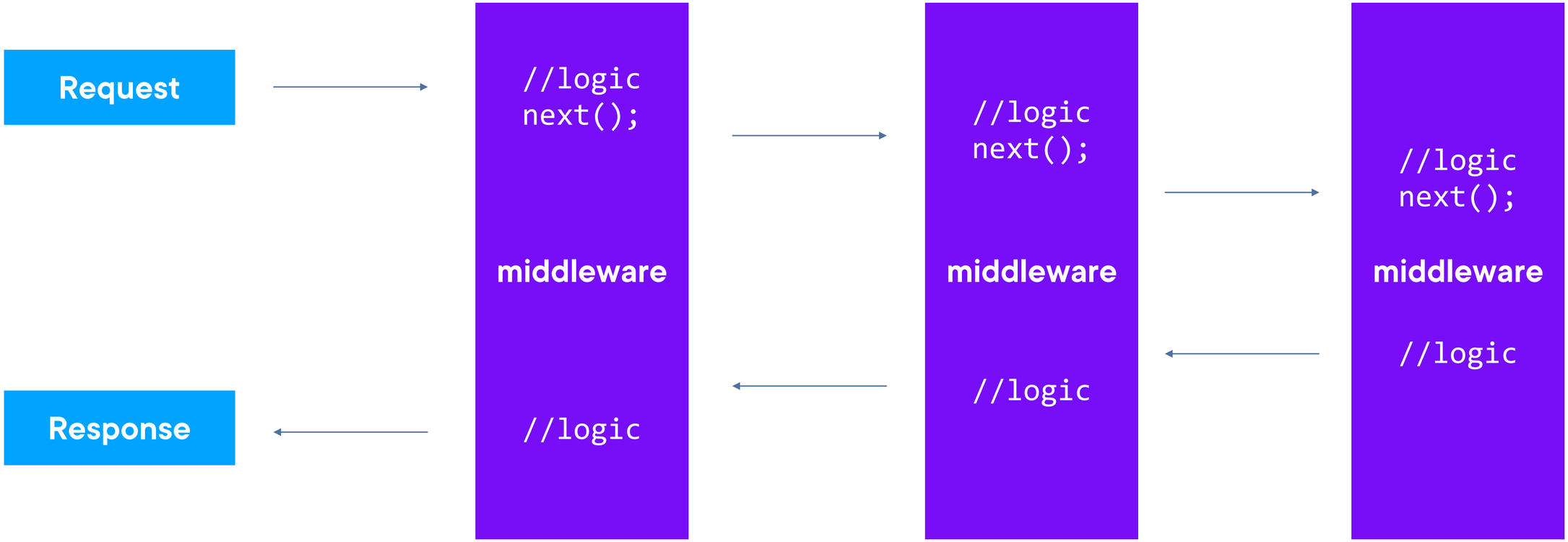
~~Mapping code in controller actions~~

Reusable `IAsyncResultFilter`

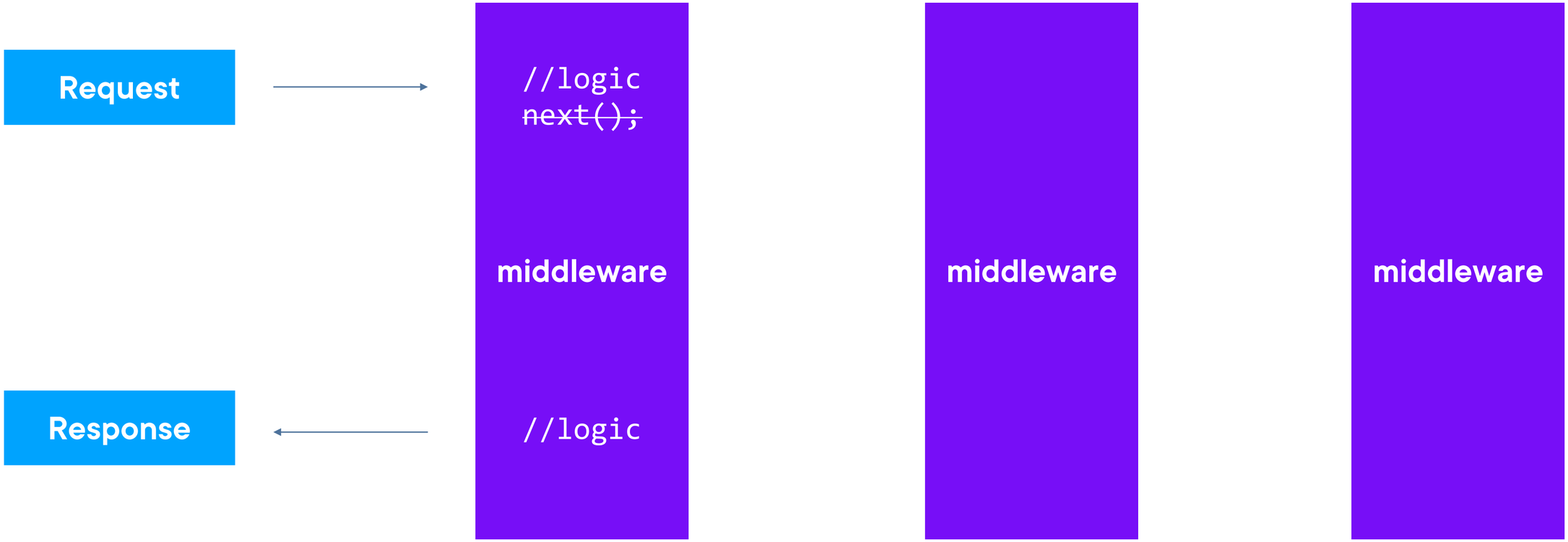
Manipulating Output with an IAsyncResultFilter

Filters in ASP.NET Core MVC allow us to run code before or after specific stages in the request processing pipeline

The ASP.NET Core Request Pipeline



The ASP.NET Core Request Pipeline

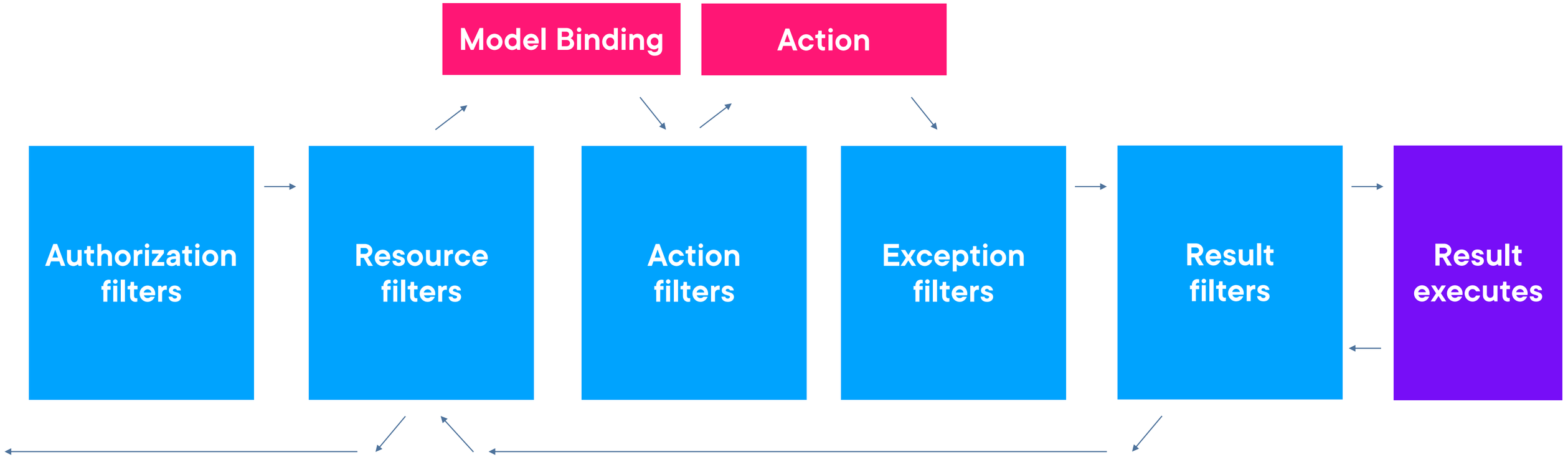


Manipulating Output with an IAsyncResultFilter

ASP.NET Core MVC has its own pipeline it sends requests through

Filters run within the MVC action invocation pipeline (aka filter pipeline)

The ASP.NET Core MVC Filter Pipeline



Manipulating Output with an IAsyncResultFilter

By using result filters, we ...

- ... can keep our actions cleaner
- ... promote reuse

`IResultFilter`, `IAsyncResultFilter`

- `ResultFilterAttribute` (abstract)

Demo

Creating a custom `IAsyncResultFilter` (part 1)

Demo

Adding and configuring AutoMapper

Demo

Creating a custom `IAsyncResultFilter` (part 2)

Summary

Async code has a tendency to bubble up application layers due to compiler errors and warnings

Summary

Keep models separate

- Mixing models & responsibilities between layers leads to evolvability issues

Summary

Result filters run right before and after the result is executed

- Makes them a good location for mapping code
- Makes the mapping code reusable

Asynchronously Manipulating Resources