Express – {Routing, Templating}

CS 390 – Web Application Development

J. Setpal

October 11, 2023

Outline

Outline

WIWYT – Query Parameters

```
https://www.youtube.com/watch?v=dQw4w9WgXcQ
                              ^------------^
```

- Parameters provide **state-specific** information about a client.

WIWYT – Query Parameters

```
https://www.youtube.com/watch?v=dQw4w9WgXcQ
                              ^------------^
```

- Parameters provide **state-specific** information about a client.
- This allows us to personalize and optimize content delivery!

WIWYT – Regex-Based Routing

- Certain behaviours share routes with content modifications (ex. `https://doma.in/user/<uid>/`; uid $= 1$ / 2 / 3 / ...).

WIWYT – Regex-Based Routing

- Certain behaviours share routes with content modifications (ex. `https://doma.in/user/<uid>/`; uid $= 1$ / 2 / 3 / ...).
- Regex-based routing allows us to minimize our program footprint.

WIWYT – Template Engines

- Speeds up writing HTML by building off a template on the fly.

WIWYT – Template Engines

- Speeds up writing HTML by building off a template on the fly.
- Enables us to serve dynamic content using server-side rendering.

Outline

Query Parameters

Each middleware prototype takes a `req` parameter as input. This request object contains further information about the query.

Query Parameters

Each middleware prototype takes a `req` parameter as input. This request object contains further information about the query.

One piece of data is the `query` object, which contains key-value pairs of data conditionally supplied to the server.

Query Parameters

Each middleware prototype takes a `req` parameter as input. This request object contains further information about the query.

One piece of data is the `query` object, which contains key-value pairs of data conditionally supplied to the server.

Example:

```
function func(req, res, next) {
        // some interesting code ...
        console.log(req.query.variable);
        // more interesting code ...
}
```

^ Logs the value of a variable with key '`variable`'.

Route Parameters

For cases wherein URL segments contain parameter information, routes can be globally specified using `:<id>` as part of a route.

Route Parameters

For cases wherein URL segments contain parameter information, routes can be globally specified using `:<id>` as part of a route.

The variable obtained from the route is parsed as a **route** parameter, and can be directly accessed from the request variable.

Route Parameters

For cases wherein URL segments contain parameter information, routes can be globally specified using `:<id>` as part of a route.

The variable obtained from the route is parsed as a **route** parameter, and can be directly accessed from the request variable.

Example:

```
app.get('/u/:uid/', (req, res) => {
        res.send(req.params.uid);
});
```

String-Based Regex

Route strings supports the following string-based regex primitives:

a. **Optional Quantifier:** `?` makes characters optional.
   Example: `targets?` accepts 'target' and 'targets'.

String-Based Regex

Route strings supports the following string-based regex primitives:

a. **Optional Quantifier:** `?` makes characters optional.
   Example: `targets?` accepts 'target' and 'targets'.

b. **Repitition Quantifier:** + checks for one or more characters.
   Example: `targets+` accepts 'targetssssss' but not 'target'.

String-Based Regex

Route strings supports the following string-based regex primitives:

a. **Optional Quantifier:** `?` makes characters optional.
   Example: `targets?` accepts 'target' and 'targets'.

b. **Repitition Quantifier:** `+` checks for one or more characters.
   Example: `targets+` accepts 'targetssssss' but not 'target'.

c. **Repitition Quantifier:** `*` checks for zero or more characters.
   Example: `targets*` accepts 'targetssssss' *and* 'target'.

String-Based Regex

Route strings supports the following string-based regex primitives:

a. **Optional Quantifier:** `?` makes characters optional.
   Example: `targets?` accepts 'target' and 'targets'.

b. **Repitition Quantifier:** `+` checks for one or more characters.
   Example: `targets+` accepts 'targetssssss' but not 'target'.

c. **Repitition Quantifier:** `*` checks for zero or more characters.
   Example: `targets*` accepts 'targetssssss' *and* 'target'.

d. **Capturing Group:** `()` groups a charset.
   Example: `print(ed)*` accepts 'print' and 'printedededededed'.

String-Based Regex

Route strings supports the following string-based regex primitives:

a. **Optional Quantifier:** `?` makes characters optional.
   Example: `targets?` accepts 'target' and 'targets'.

b. **Repitition Quantifier:** `+` checks for one or more characters.
   Example: `targets+` accepts 'targetssssss' but not 'target'.

c. **Repitition Quantifier:** `*` checks for zero or more characters.
   Example: `targets*` accepts 'targetssssss' *and* 'target'.

d. **Capturing Group:** `()` groups a charset.
   Example: `print(ed)*` accepts 'print' and 'printedededededed'.

These primitives can be included as part of the route string.
Example:

```
app.get('/targets?', (req, res) => { res.send(req.url); });
```

Regex-Only Rules

For *pure* regex rules, use the JS Regex with `/<exp>/` instead of string route input.

Regex-Only Rules

For *pure* regex rules, use the JS Regex with `/<exp>/` instead of string route input.

Example: `app.get(/[A-Za-z0-9]+, [A-Za-z0-9]+!/), middleware);`

Regex-Only Rules

For *pure* regex rules, use the JS Regex with `/<exp>/` instead of string route input.

Example: `app.get(/[A-Za-z0-9]+, [A-Za-z0-9]+!/), middleware);`
^ Matches syntax based on two comma-separated words followed by '!'.

Regex-Only Rules

For *pure* regex rules, use the JS Regex with `/<exp>/` instead of string route input.

Example: `app.get(/[A-Za-z0-9]+, [A-Za-z0-9]+!/), middleware);`
^ Matches syntax based on two comma-separated words followed by '!'.

Regex Generator: `https://regex-generator.olafneumann.org/`

Route Error Handling

By default, Express handles errors by returning the error message (and stack trace, if not deployed in production) to the client.

Route Error Handling

By default, Express handles errors by returning the error message (and stack trace, if not deployed in production) to the client.

This behaviour can be customized by specifying a function with prototype `(err, req, res, next)` and integrating it to the api with `app.use`.

Route Error Handling

By default, Express handles errors by returning the error message (and stack trace, if not deployed in production) to the client.

This behaviour can be customized by specifying a function with prototype `(err, req, res, next)` and integrating it to the api with `app.use`.

We can specify response status code with `res.sendStatus(<code>)` or `res.status(<code>).send(<message>)` with a message.

Route Error Handling

By default, Express handles errors by returning the error message (and stack trace, if not deployed in production) to the client.

This behaviour can be customized by specifying a function with prototype `(err, req, res, next)` and integrating it to the api with `app.use`.

We can specify response status code with `res.sendStatus(<code>)` or `res.status(<code>).send(<message>)` with a message.

Example:

```
app.use((err, req, res, next) => {
        console.error(err.stack);
        res.status(500).send('Something went wrong :/');
});
```

Let's Implement (Naive) Auth!

If you can view this screen, I am making a mistake.

Outline

What's a Template Engine?

Template Engines are used to ease and automate writing HTML.

What's a Template Engine?

Template Engines are used to ease and automate writing HTML.

There are several popular engines - today we'll be looking at **pug**: `https://pugjs.org/api/getting-started.html`.

What's a Template Engine?

Template Engines are used to ease and automate writing HTML.

There are several popular engines - today we'll be looking at **pug**: `https://pugjs.org/api/getting-started.html`.

It uses a markdown-like syntax. Has features like conditions, loops, includes & mixins.

Pug Syntax

Each element is **only defined once**.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.

b. `#<var>` after the element specifies the element id.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.

b. `#<var>` after the element specifies the element id.

c. `.<var>` after the element specifies the element class(es).

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.
b. `#<var>` after the element specifies the element id.
c. `.<var>` after the element specifies the element class(es).
d. Elements can be made multiline using a '`.`' at the end of the element.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.
b. `#<var>` after the element specifies the element id.
c. `.<var>` after the element specifies the element class(es).
d. Elements can be made multiline using a '`.`' at the end of the element.
e. Javascript can be injected using '`-`' at the beginning of the line.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.
b. `#<var>` after the element specifies the element id.
c. `.<var>` after the element specifies the element class(es).
d. Elements can be made multiline using a '`.`' at the end of the element.
e. Javascript can be injected using '`-`' at the beginning of the line.
f. Attributes can be specified using `elem(key="val" key2="val")`.

Pug Syntax

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.
b. `#<var>` after the element specifies the element id.
c. `.<var>` after the element specifies the element class(es).
d. Elements can be made multiline using a '`.`' at the end of the element.
e. Javascript can be injected using '`-`' at the beginning of the line.
f. Attributes can be specified using `elem(key="val" key2="val")`.

These are each included in a `.pug` file. This generates HTML output that can be rendered on-the-fly or statically.

Loops

Pug can loop through an array or object to procedurally render elements.

Loops

Pug can loop through an array or object to procedurally render elements.

Syntax:

```
for/each <var> in <array/object>
        <elem>= <var>
        ... additional interesting code
```

Example:

```
ul
   for i in [0, 1, 2, 3]
        li= i
```

Loops

Pug can loop through an array or object to procedurally render elements.

Syntax:

```
for/each <var> in <array/object>
        <elem>= <var>
        ... additional interesting code
```

Example:

```
ul
   for i in [0, 1, 2, 3]
        li= i
```

The input array can be specified dynamically by supplying a variable through express.

Loops

Pug can loop through an array or object to procedurally render elements.

Syntax:

```
for/each <var> in <array/object>
        <elem>= <var>
        ... additional interesting code
```

Example:

```
ul
   for i in [0, 1, 2, 3]
        li= i
```

The input array can be specified dynamically by supplying a variable through express.

`else` can be used to specify default behavior when no items are present to iterate through.

Conditionals #1

Pug implements if/else and switch statements to conditionally render elements.

Conditionals #1

Pug implements if/else and switch statements to conditionally render elements.

Syntax:

```
if <condition>
        ... stuff to render
else if <condition>
        ... stuff to render
else <condition>
        ... stuff to render
```

Example:

```
- const book = {genre: "horror", fiction: true}
if book.fiction
        p= book.genre
```

Conditionals #2

Switch is helpful when evaluating categorical values.

Conditionals #2

Switch is helpful when evaluating categorical values.

Syntax:

```
case <var>
        when <value>
                ... stuff to render
        when <value>
                ... stuff to render
```

Example:

```
- const book = {genre: "horror", fiction: true, rating: 10}
case book.genre
        when "horror"
                p= book.rating
        when "sci-fi"
                strong 10/10 best book ever
```

Includes

Pug, fundamentally, is build around the idea of *minimizing* how much we type.

Therefore, it integrates **includes** and **mixins** to follow DRY.

Includes

Pug, fundamentally, is build around the idea of *minimizing* how much we type.

Therefore, it integrates **includes** and **mixins** to follow DRY.

Includes are static renderable chunks of templates, that can be re-used in various template files.

Includes

Pug, fundamentally, is build around the idea of *minimizing* how much we type.

Therefore, it integrates **includes** and **mixins** to follow DRY.

Includes are static renderable chunks of templates, that can be re-used in various template files.

They are added using `include /path/to/file.pug`.

Mixins

Mixins are cross between *functions* and *includes*.

Mixins

Mixins are cross between *functions* and *includes*.

You can specify the location at which a chunk is rendered, similar to includes. However; unlike includes, mixins are **not restricted to static data**.

Mixins

Mixins are cross between *functions* and *includes*.

You can specify the location at which a chunk is rendered, similar to includes. However; unlike includes, mixins are **not restricted to static data**.

Syntax:

```
mixin func(var1, var2)
        p= var1 + ' and ' + var2

+func('Hello', 'World')
```

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.

b. Set this folder as '`views`' using `app.set`.

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.
b. Set this folder as '`views`' using `app.set`.
c. Set '`view engine`' as '`pug`' using `app.set`.

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.
b. Set this folder as '`views`' using `app.set`.
c. Set '`view engine`' as '`pug`' using `app.set`.
d. In the route response, specify `res.render(<file>, {<key>: <value>});` to dynamically render the file server-side.

Let's Build a File Host Frontend!

If you can view this screen, I am making a mistake (again).

Outline

Thank you!

Have an awesome rest of your day!

Slides: `https://www.cs.purdue.edu/homes/jsetpal/slides/routing,templating.pdf`

If anything's incorrect or unclear, please ping: `jsetpal@purdue.edu`
I'll patch it ASAP.