

# THUNK MIDDLEWARE

*Who'da thunk it?*

# TRAJECTORY

- **What the thunk?**
- **Why the thunk would I do this?**
- **When to thunk?**


# WHAT IS THUNK MIDDLEWARE?

- **Checks the incoming action**
  - **If the action is a regular object, do nothing**
  - **If the “action” is a function, invoke it, and pass the store’s dispatch and getState methods to it!**
  - **We call that function a “thunk” (it’s just a term borrowed from functional programming and elsewhere)**

Branch: master ▾

redux-thunk / src / index.js

Fi

 **gareon** Add withExtraArgument()

41aef

2 contributors



15 lines (11 sloc) 352 Bytes

Raw

Blame


History

```
1  function createThunkMiddleware(extraArgument) {
2    return ({ dispatch, getState }) => next => action => {
3      if (typeof action === 'function') {
4        return action(dispatch, getState, extraArgument);
5      }
6
7      return next(action);
8    };
9  }
10
11  const thunk = createThunkMiddleware();
12  thunk.withExtraArgument = createThunkMiddleware;
13
14  export default thunk;
```

Branch: master ▾

redux-thunk / src / index.js

Fi

 gaearon Add withExtraArgument()

41aef

2 contributors




15 lines (11 sloc) 352 Bytes

Raw

Blame

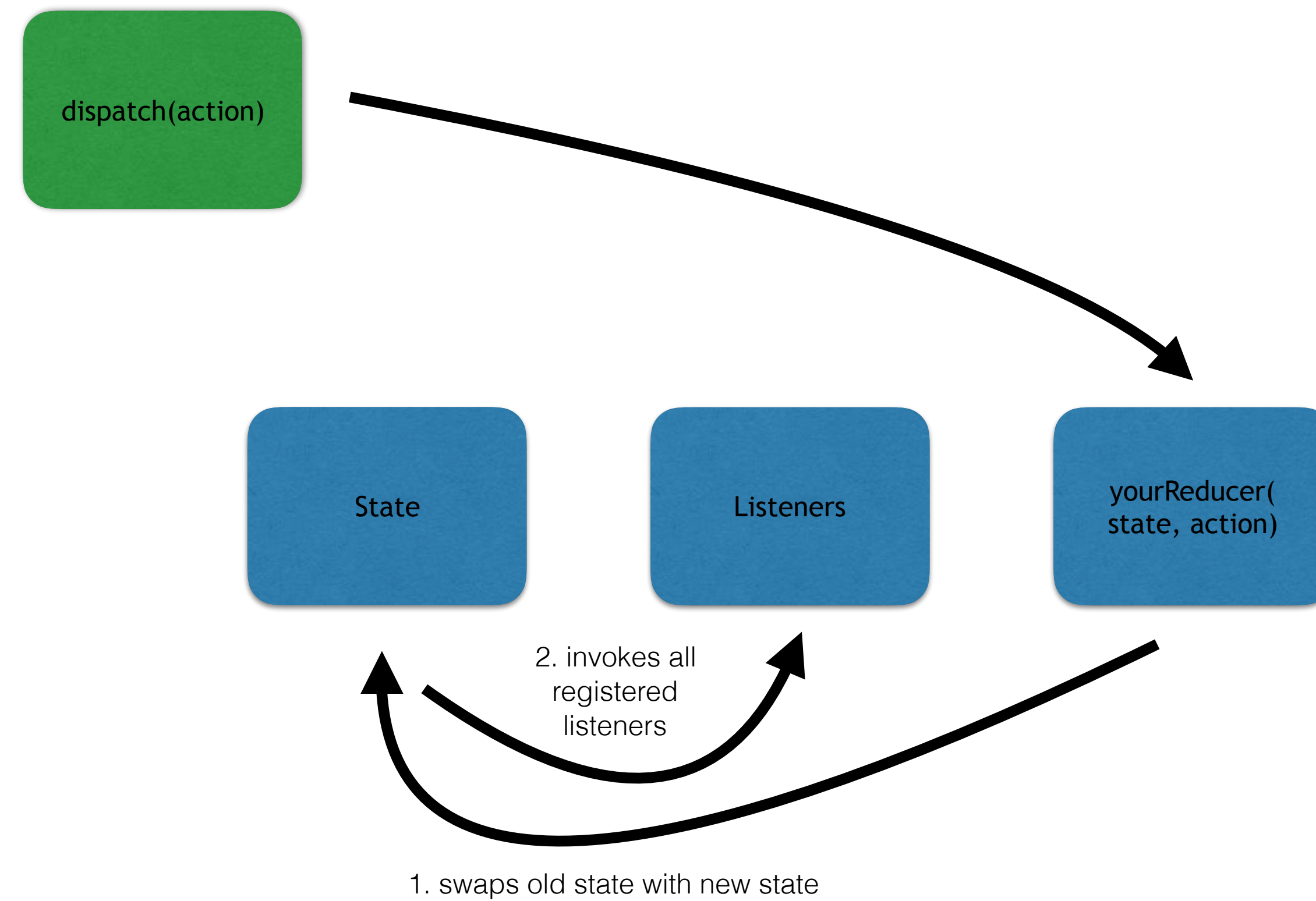
History

```
1  function createThunkMiddleware(extraArgument) {
2    return ({ dispatch, getState }) => next => action => {
3      if (typeof action === 'function') {
4        return action(dispatch, getState, extraArgument);
5      }
6
7      return next(action);
8    };
9  }
10
11  const thunk = createThunkMiddleware();
12  thunk.withExtraArgument = createThunkMiddleware;
13
14  export default thunk;
```

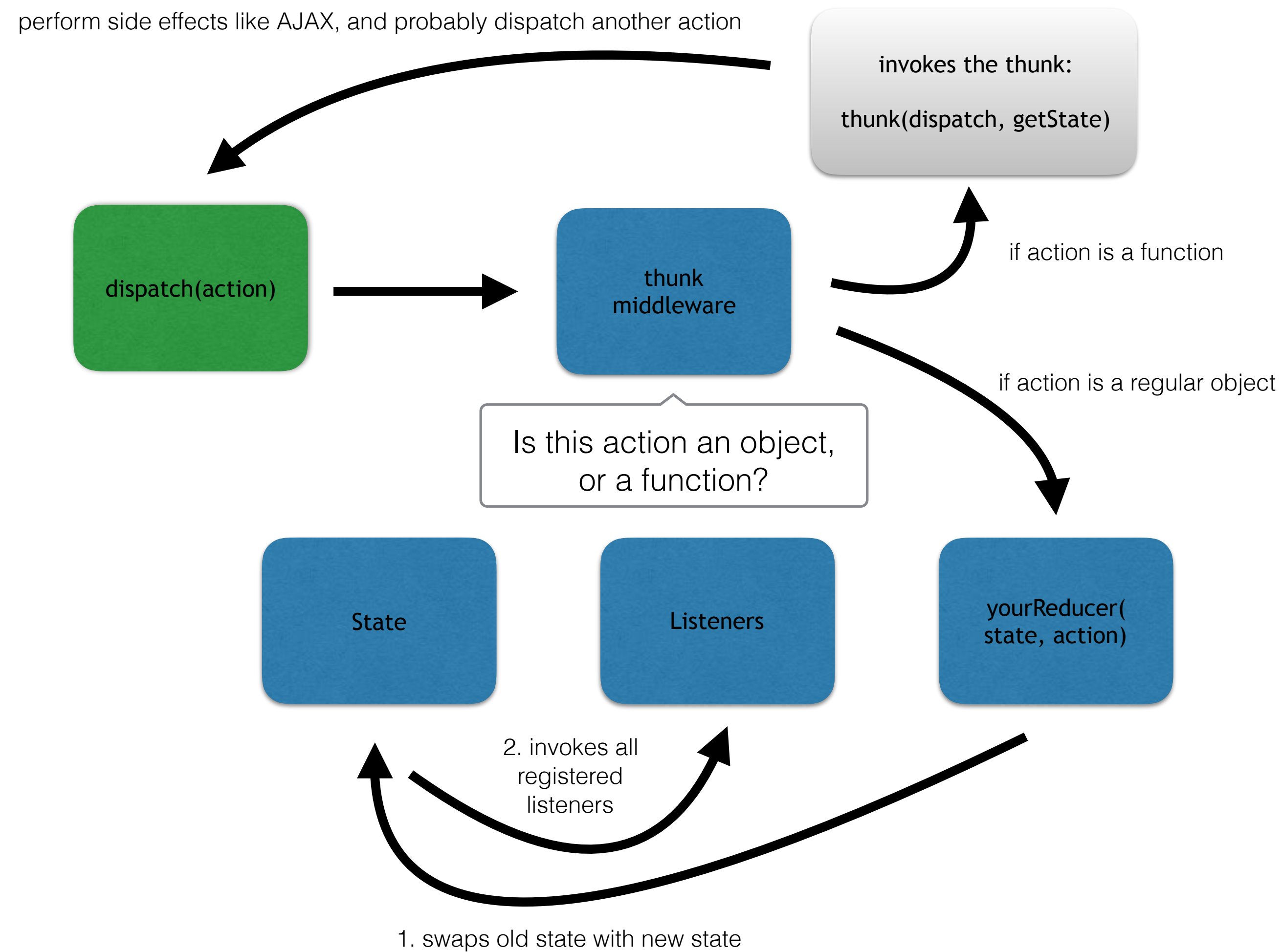




# WITHOUT THUNK MIDDLEWARE



# WITH THUNK MIDDLEWARE



# WHY THINK?

- **Helps you stay DRY**
  - Several different components might want to make the same API requests.
- **Removes “responsibility” for asynchronous code/side effects from components**
  - Components don’t need to know whether an action is async or not - they just dispatch!



# TERMINOLOGY

- **Thunk creator: an action creator that returns a thunk; also known as “async action creators”**
- **Thunk: the function returned from a thunk creator; it accepts *dispatch* and *setState* as arguments**

```
const GET_PUGS = 'GET_PUGS'
```

```
const getPugs = (pugs) => {
```

```
  return {  
    type: GET_PUGS,  
    pugs  
  }  
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

const getPugs = (pugs) => {

  return {
    type: GET_PUGS,
    pugs
  }
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

// the "action creator"
const getPugs = (pugs) => {

  return {
    type: GET_PUGS,
    pugs
  }
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

// the "action creator"
const getPugs = (pugs) => {
  // the "action"
  return {
    type: GET_PUGS,
    pugs
  }
}
```



```
// the "action type"  
const GET_PUGS = 'GET_PUGS'
```

```
// the "action creator"  
const getPugs = (pugs) => {  
  // the "action"  
  return {  
    type: GET_PUGS,  
    pugs  
  }  
}
```

```
const getPugs = () => {  
  
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

// the "action creator"
const getPugs = (pugs) => {
  // the "action"
  return {
    type: GET_PUGS,
    pugs
  }
}
```

```
const getPugs = () => {
  return async (dispatch, getState) => {
    }
  }
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

// the "action creator"
const getPugs = (pugs) => {
  // the "action"
  return {
    type: GET_PUGS,
    pugs
  }
}
```

```
const getPugs = () => {
  return async (dispatch, getState) => {
    const {data} = await axios.get('/api/pugs')
    dispatch(getPugs(data))
  }
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

// the "action creator"
const getPugs = (pugs) => {
  // the "action"
  return {
    type: GET_PUGS,
    pugs
  }
}
```

```
// the "thunk creator"
const getPugs = () => {
  return async (dispatch, getState) => {
    const {data} = await axios.get('/api/pugs')
    dispatch(getPugs(data))
  }
}
```

```
// the "action type"
const GET_PUGS = 'GET_PUGS'

// the "action creator"
const getPugs = (pugs) => {
  // the "action"
  return {
    type: GET_PUGS,
    pugs
  }
}
```

```
// the "thunk creator"
const getPugs = () => {
  // the "thunk"
  return async (dispatch, getState) => {
    const {data} = await axios.get('/api/pugs')
    dispatch(getPugs(data))
  }
}
```



```
import React from 'react'  
import {connect} from 'react-redux'  
import {getPugs} from '../store'
```

```
import React from 'react'
import {connect} from 'react-redux'
import {getPugs} from '../store'

const AllPugs = connect(

)

)
```

```
import React from 'react'
import {connect} from 'react-redux'
import {getPugs} from '../store'

const AllPugs = connect(

)(class extends React.Component {

})
```

```
import React from 'react'
import {connect} from 'react-redux'
import {getPugs} from '../store'

const AllPugs = connect(
  // mapStateToProps
  (state) => {
    return {
      pugs: state.pugs
    }
  },

)(class extends React.Component {

})
```

```
import React from 'react'
import {connect} from 'react-redux'
import {getPugs} from '../store'

const AllPugs = connect(
  // mapStateToProps
  (state) => {
    return {
      pugs: state.pugs
    }
  },
  // mapDispatchToProps
  (dispatch) => ({
    getPugs: () => dispatch(getPugs())
  })
)(class extends React.Component {

})
```



```
import React from 'react'
import {connect} from 'react-redux'
import {getPugs} from '../store'

const AllPugs = connect(
  // mapStateToProps
  (state) => {
    return {
      pugs: state.pugs
    }
  },
  // mapDispatchToProps
  (dispatch) => ({
    getPugs: () => dispatch(getPugs())
  })
)(class extends React.Component {
  componentDidMount () {
    this.props.getPugs()
  }
  // etc....
})
```

# MUST I ALWAYS THUNK?

- **No, not always**
- **Thunks, like Redux itself, are tools to help organize big applications**
  - **Sitting down to write your company's big analytics dashboard app? Thunk it up!**
  - **Writing something fun for your personal site? Not the time to be thunky. Not even the time for Redux!**