Forte

Schema-driven React form engine

Controlled Component

> In HTML, form elements such as `<input>`, `<textarea>`, and `<select>` typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with setState().
> We can combine the two by making the React state be the “single source of truth”. Then the React component that renders a form also controls what happens in that form on subsequent user input.

Read more

Controlled Component

```
const Field = ({ name,  children }) => {
  const form = React.useContext(FormContext)
  const { value, onChange } = form.control(name)
  return React.cloneElement(children, { value, onChange })
}
```

NATIVE CONTROLLED COMPONENT

```
const Search = ({ onSearch }) => {
  const [keyword, setKeyword] = React.useState('')
  const handleSubmit = (e) => {
    e.preventDefault()
    onSearch({ keyword })
  }
  return (
    <form onSubmit={handleSubmit}>
      <input
        value={keyword}
        onChange={e => setKeyword(e.target.value)} />
    </form>
  )
}
```

RC-FIELD-FORM

```
import { Form, Field } from 'rc-field-form'

const Search = ({ onSearch }) => {
  const handleFinish = ({ keyword }) =>
    onSearch({ keyword })
  return (
    <Form onFinish={handleFinish}>
      <Field name="keyword" rules={rules}>
        <input />
      </Field>
    </Form>
  )
}
```

Problems encountered in real-world scenarios

```
export const ServiceForm = () => {
  return (
    <Form>
      {(form) => {
        const isHealthCheckEnabled = form.getFieldValue(['healthCheck', 'isEnabled'])
        return <>
          <Field name={['healthCheck', 'isEnabled']} label="心跳上报">
            <Switch />
          </Field>
          {isHealthCheckEnabled && (
            <Field name={['healthCheck', 'ttl']} label="TTL (秒)">
              <Input placeholder="请输入心跳上报 TTL 秒数" />
            </Field>
          )}
        </>
      }}
    </Form>
  )
}
```

Render-Props will slow down rendering as the number of child nodes increases.

Problems encountered in real-world scenarios

```
export const ServiceForm = () => {
  const [form] = useForm()
  const isHealthCheckEnabled = form.getFieldValue(['healthCheck', 'isEnabled'])
  return (
    <Form form={form}>
      <Field name={['healthCheck', 'isEnabled']} label="心跳上报">
        <Switch />
      </Field>
      {isHealthCheckEnabled && (
        <Field name={['healthCheck', 'ttl']} label="TTL (秒)">
          <Input placeholder="请输入心跳上报 TTL 秒数" />
        </Field>
      )}
    </Form>
  )
}
```

Using the hoisted context to access the form instance won’t help either.

Problems encountered in real-world scenarios

```
export const ServiceForm = ({ services }: { services: IService[] }) => {
  const rules = React.useMemo(() => [
    () => ({
      validator: async (_, name) => {
        if (!services) {
          return;
        }
        assert(services?.every(service => service?.name !== name), `已存在名称为 ${name} 的 Service`);
      },
    }),
  ], [services]);
  return (
    <Form>
      <Field name="name" label="服务名称" rules={rules}>
        <Input type="text" placeholder="请输入服务名称" />
      </Field>
    </Form>
  )
};
```

How to trigger validation that depends on the state outside the form when the dependency changes?

Problems encountered in real-world scenarios

```
export const ServiceForm = () => {
  const [form] = useForm()
  return <Form form={form}>
    <BasicForm form={form} />
    <HealthCheckForm form={form} />
  </Form>
}

export const HealthCheckForm = ({ form }: { form: FormInstance }) => {
  const isEnabled = form.getFieldValue(['healthCheck', 'isEnabled'])
  return <Body>
    <Field name={['healthCheck', 'isEnabled']} label="心跳上报">
      <Switch />
    </Field>
    <Field name={['healthCheck', 'ttl']} label="TTL (秒)">
      <Input placeholder="请输入心跳上报 TTL 秒数" />
    </Field>
  </Body>
}
```

Can we omit redundant prefix paths in nested components?

What is Forte?

Forte is a Schema-driven React form engine, designed for decoupling and componentization.

- 🧩 Schema Driven
- 🏎️ Performance First
- 📏 Validation
- 👯 Efficient List
- 🪆 Scope Componentization
- 🪝 React Hooks Integration
- 💭 Type Infering

Basic Usage

```
import { Form, Field, Schema as S } from '@fortejs/forte'

const FormSchema = S.Form({
  username: S.Field<string>(),
  password: S.Field<string>(),
})

export const App = () => {
  const handleSubmit = React.useCallback(values => console.log(values), [])
  return (
    <>
      <h3>Login</h3>
      <Form schema={FormSchema} onSubmit={handleSubmit}>
        <Field path="username">{control => <input placeholder="Username" {...control} />}</Field>
        <Field path="password">{control => <input placeholder="Password" type="password" {...control} />}</Field>
        <input type="submit" />
      </Form>
    </>
  )
}
```

Componentization

```
import { FormScope, Field, S } from '@fortejs/forte'
import { PolarisFormSchema, PolarisForm } from './polaris'

const ServiceFormSchema = S.Form({
  name: S.Field<string>(),
  polaris: PolarisFormSchema,
})

export const ServiceForm = () => {
  const handleSubmit = React.useCallback(values =>
    console.log(values)
  , [])
  return (
    <Form schema={ServiceFormSchema} onSubmit={handleSubmit
      <Field path="name">{control =>
        <input placeholder="service name" {...control} />
      }</Field>
      <FormScope path="polaris">
        <PolarisForm />
      </FormScope>
    </Form>
  )
}
```

```
import { Field, S } from '@fortejs/forte'

export const PolarisFormSchema = S.Scope({
  name: S.Field<string>(),
  token: S.Field<string>(),
})

export const PolarisForm = () => {
  return (
    <>
      <Field path="name">{control =>
        <input placeholder="polaris name" {...control} />
      }</Field>
      <Field path="token">{control =>
        <input placeholder="polaris token" {...control} />
      }</Field>
    </>
  )
}
```

Validation with builtin predicates

```
export const ServiceFormSchema = S.Scope({
  name: S.Field<string>({
    defaultValue: '',
    rules: [
      ['string/required', []],
      ['string/max', [1000]],
      ['string/pattern', [/^[a-z]([-a-z0-9]*[a-z0-9])?$/]],
    ],
  }),
})
```

Validation with dependencies

```
export const ServiceFormSchema = S.Scope({
  name: S.Field<string, [IService[], INamespace]>({
    defaultValue: '',
    rules: [
      { predicate: ['string/required', []], lazy: true },
      { predicate: ['string/max', [1000], lazy: true },
      { predicate: ['string/pattern', [/^[a-z]([-a-z0-9]*[a-z0-9])?$/]], lazy: true },
      async (value, [services, namespace]) => {
        assert(
          !services?.some(
            service =>
              service?.name === value &&
              service?.namespace?.name === namespace?.name &&
              service?.namespace?.cluster?.id === namespace?.cluster?.id
          ),
          `同集群同命名空间下已存在名称为 ${value} 的 Service`
        )
      },
    ],
  }),
})
```

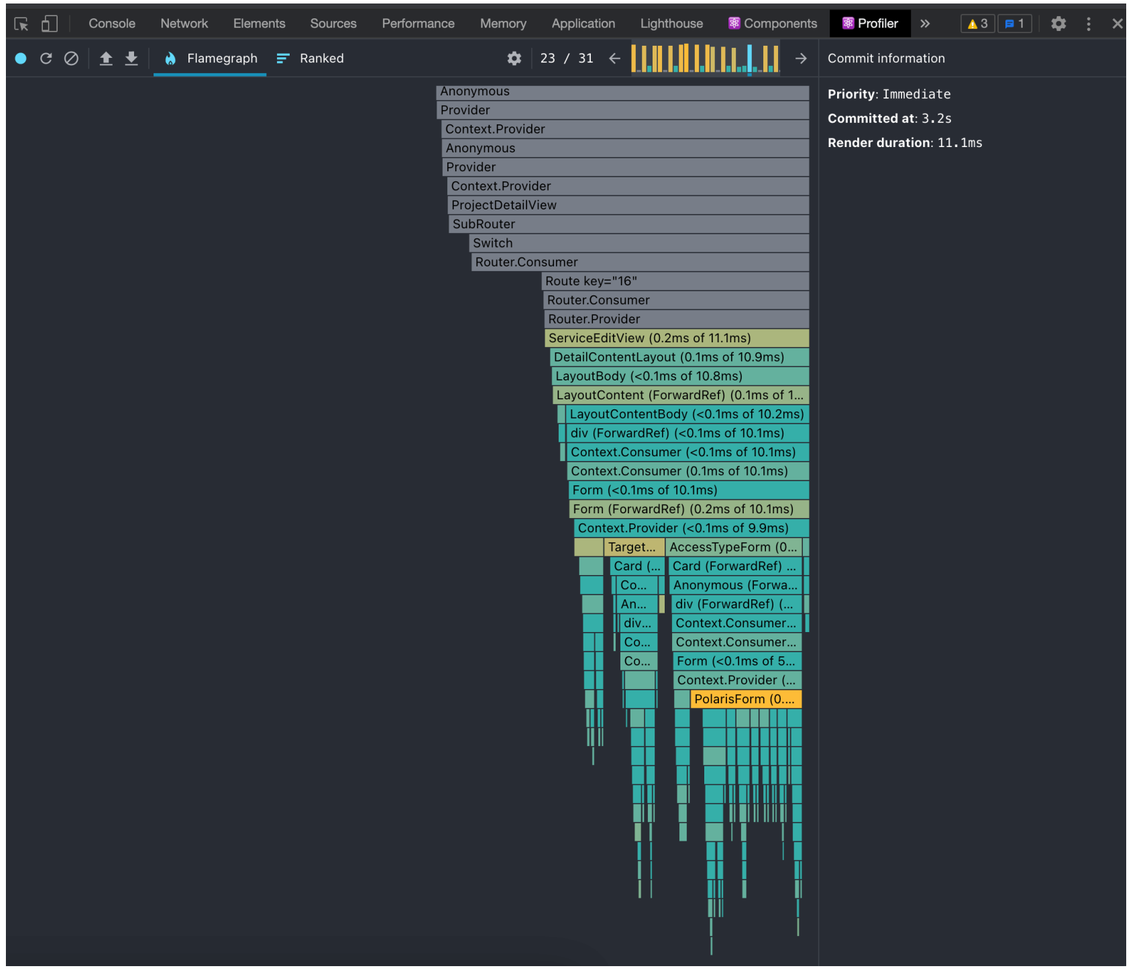
Validation with dependencies

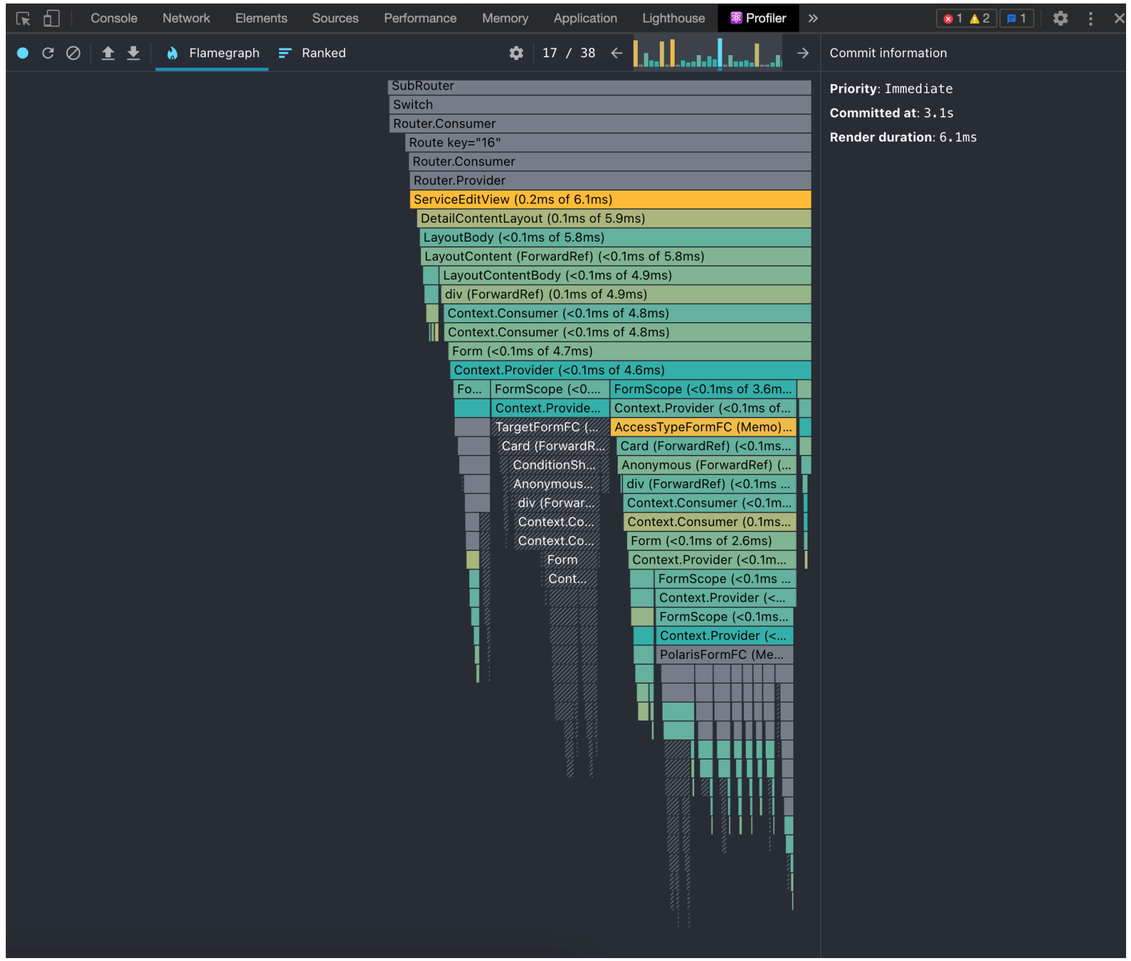
```
import { Form, Field } from '@fortejs/forte'

const ServiceForm = ({ namespace }: { namespace: INamespace }) => {
  const { services } = React.useContext(ServicesContext)

  return (
    <FormScope>
      <Field path="name" dependencies={[services, namespace]}>
        <Input type="text" placeholder="请输入服务名称" />
      </Field>
    </FormScope>
  )
}
```

Validation with dependencies

```
import { Form, Field, useForteValue } from '@fortejs/forte'

export const ServiceFormSchema = S.Scope({
  name: S.Field<string, [IService[], INamespace]>({ /** ... */ }),
  namespace: S.Field<INamespace>({ /** ... */ }),
})

const ServiceForm = () => {
  const { services } = React.useContext(ServicesContext)
  const namespace = useForteValue('namespace')

  return (
    <FormScope>
      <Field path="name" dependencies={[services, namespace]}>
        <Input type="text" placeholder="请输入服务名称" />
      </Field>
      <Field path="namespace">
        <NamespaceSelect />
      </Field>
    </FormScope>
  )
}
```

Using Hooks with Subscription



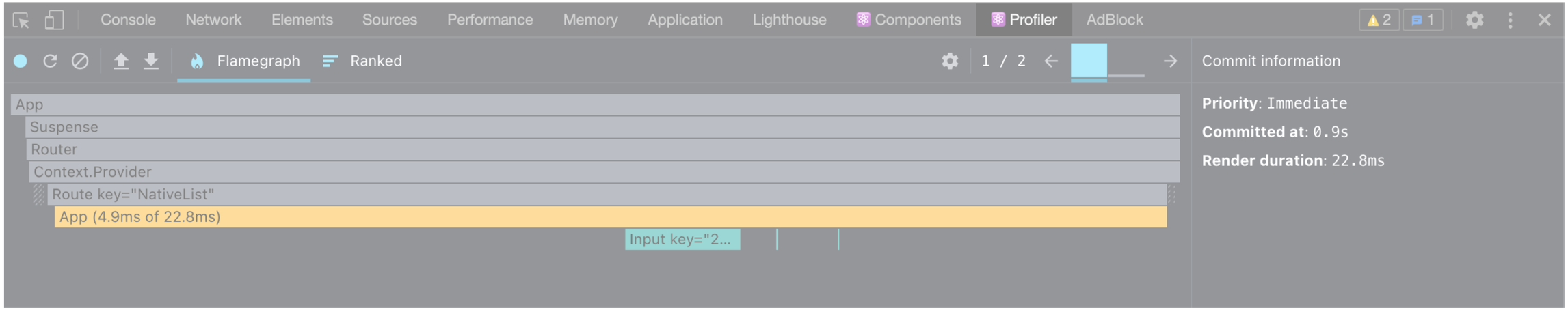


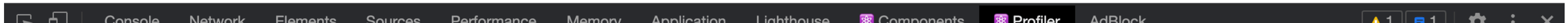
Big lists are dangerous

```
const App = () => {
  const [names, setNames] = React.useState<string[]>([])
  const setName = React.useCallback((value: string, index: number) => {
    setNames([...names.slice(0, index), value, ...names.slice(index + 1)])
  }, [names])
  return (
    <>
      {names.map((name, index) => (
        <Input key={String(index)} value={name} onChange={value => setName(value, index)} />
      ))}
    </>
  )
}
```

Since `setName` functions always change with any names change, using key prop does not reduce redundant re-rendering.

List Hijacking





List Usage

```
import { Form, Field, FormList, S } from '@fortejs/forte'

const FormSchema = S.Form({
  tags: S.List({ name: S.Field<string>({ defaultValue: '' }) }),
})

const TagForm = React.memo(() => <Field path="name">{control => <input placeholder="name" {...control} />}</Field>)

export const App = () => {
  const handleSubmit = React.useCallback(values => console.log('submit', values), [])
  return (
    <Form schema={FormSchema} onSubmit={handleSubmit}>
      <FormList path="tags">
        {({ map, push }) => (
          <>
            {map(() => <TagForm />)}
            <button type="button" onClick={() => push({ name: '' })}>+ Add</button>
          </>
        )}
      </FormList>
      <button type="submit">Submit</button>
    </Form>
  )
}
```

No keys requierd

- In a UI, it’s not necessary for every update to be applied immediately; in fact, doing so can be wasteful, causing frames to drop and degrading the user experience.
- Different types of updates have different priorities — an animation update needs to complete more quickly than, say, an update from a data store.
- A push-based approach requires the app (you, the programmer) to decide how to schedule work. A pull-based approach allows the framework (React) to be smart and make those decisions for you.

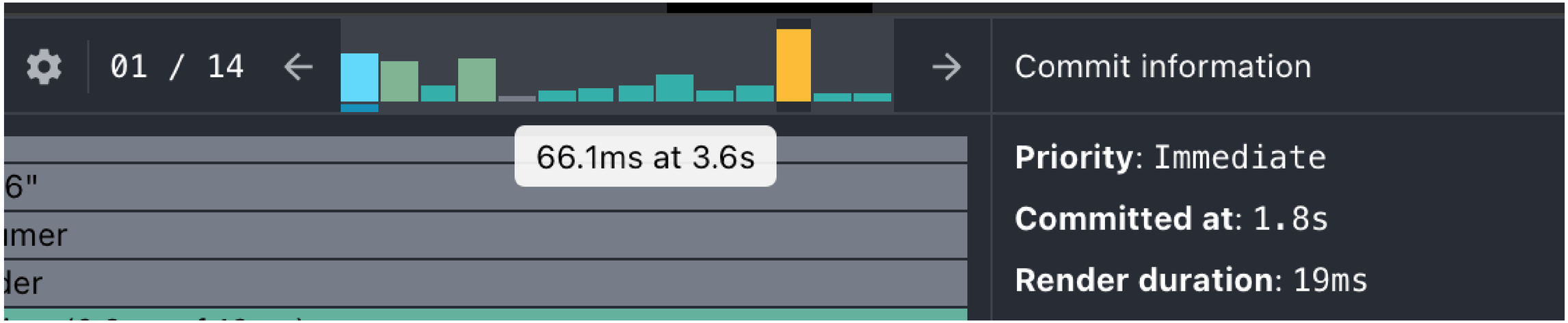
React Fiber Architecture - Scheduling

> Mostly backwards compatibility reasons. The Node.js team can't break the whole ecosystem.
>
> It also allows silly code like this:
>
> ```
> let unicorn = false;
>
> emitter.on('🦄', () => {
>  unicorn = true;
> });
>
> emitter.emit('🦄');
>
> console.log(unicorn);
> //=> true
> ```
>
> But I would argue doing that shows a deeper lack of Node.js and async comprehension and is not something we should optimize for. The benefit of async emitting is much greater.

sindresorhus/emittery - Isn't EventEmitter synchronous for a reason?

EventEmitter, Sync or Async

sindresorhus/emittery



Type Infering

TypeScript 4.1 - Template Literal Types

```
168   }, []);
169
170   React.useEffect(() => {
171     const workloadType =
172       WORKLOAD_TYPE_OPTION_LOWERCA
173     if (workloadType) {
174       setter('value/workoadType', workloadType);
175     }
176     setter('value/workloadName', selectors?.find(selector => selector.key === 'qcloud-app')?.value);
177   }, [setter, selectors]);
178
```

```
const workloadType: TWorkloadType

Argument of type 'string' is not assignable to parameter of type 'never'.
  Type 'string' is not assignable to type 'never'. ts(2345)

View Problem (⌥F8)    No quick fixes available
```

Which one should I choose?

Redux(-like)

PROS

- 🏎️ Good Performance

RC-Field-Form(-like)

PROS

- 🎨 Designed for Form

Forte

PROS

- 🏎️ Good Performance
- 🎨 Designed for Form
- 🪆 Scope Componentization
- 💭 TypeScript Support

CONS

- 🧱 Schema Required

Example

命名空间 ns-prjtrtzv-1081093-production @ 集群 cls-l2xpq2bd (大核心-TKEX-Hui

Selectors

clusterId	=	cls-l2xpq2bd
env	=	test
k8s-app	=	init-container
moduleFourId	=	1081093
projectName	=	prjtrtzv
qcloud-app	=	init-container
region	=	ap-tianjin
worload-kind	=	statefulsetplus

添加

TKEx-CSIG

北极星 / L5 / CL5 / CMLB 类型 Service 请确保包含 Selector 字段: qcloud-app, k8s-app, workload-type

访问设置 (Service)

访问方式: 北极星 | L5 路由 | CL5 路由 | CMLB 路由 | 集群内访问 (ClusterIP) | 主机端口访问 (NodePort) | 内网访问 (四层转发) | 公网访问 (四层转发)

将提供一个可以从 Internet 访问入口，支持 TCP/UDP 协议，如 Web 前台类服务可以选择公网访问。如您需要公网通过 HTTP/HTTPS 协议或根据 URL 转发，您可以在 Ingress 页面使用 Ingress 进行路由转发。

负载均衡器: 自动创建 | 使用已有

自动创建 CLB 用于公网/内网访问 Service，请勿手动修改由 TKE 创建的 CLB 监听器，查看更多说明

VIP 运营商 ⓘ *: 电信 | 联通 | 移动 | BGP

限速 ⓘ: 限速 CAP

Roadmap

- 💂 Higher test coverage *(currently 85%)*
- 🚌 New EventEmitter Provider
- 📖 Better Documents
- 💭 Better Type Infering

Learn More

- Repo
- Document (WIP)