graphql序列化请求

Why 序列化

persist可以在服务端缓存你的请求体，在每次查询的时候不必发送相同的查询，这样可以减少网络带宽

如何实现序列化

我们利用边缘缓存来避免不必要的请求到服务器,允许拦截请求并在早期从内存返回，所以我们也想利用GraphQL序列化请求为节省带宽。自GraphQL发送POST请求到一个端点,边缘缓存就很难完成。这就是GET请求可以帮助我们。

example

- 创建一个schema和resolver文件

```
touch schemas.js resolvers.js
```

- schemas.js

```
const Greeting = `
  type Greeting {
    name: String
    text: String
  }
`
const Query = `
  type Query {
    greeting(name: String): Greeting
  }
`
module.exports = [Greeting, Query]
```

- resolve.js

```
const resolvers = {
  Query: {
    greeting: (_, { name }) => ({
      name,
      text: 'How are you today?'
    })
  }
}
module.exports = resolvers
```

- 然后我们创建一个extracted queries file: touch extracted_queries.js

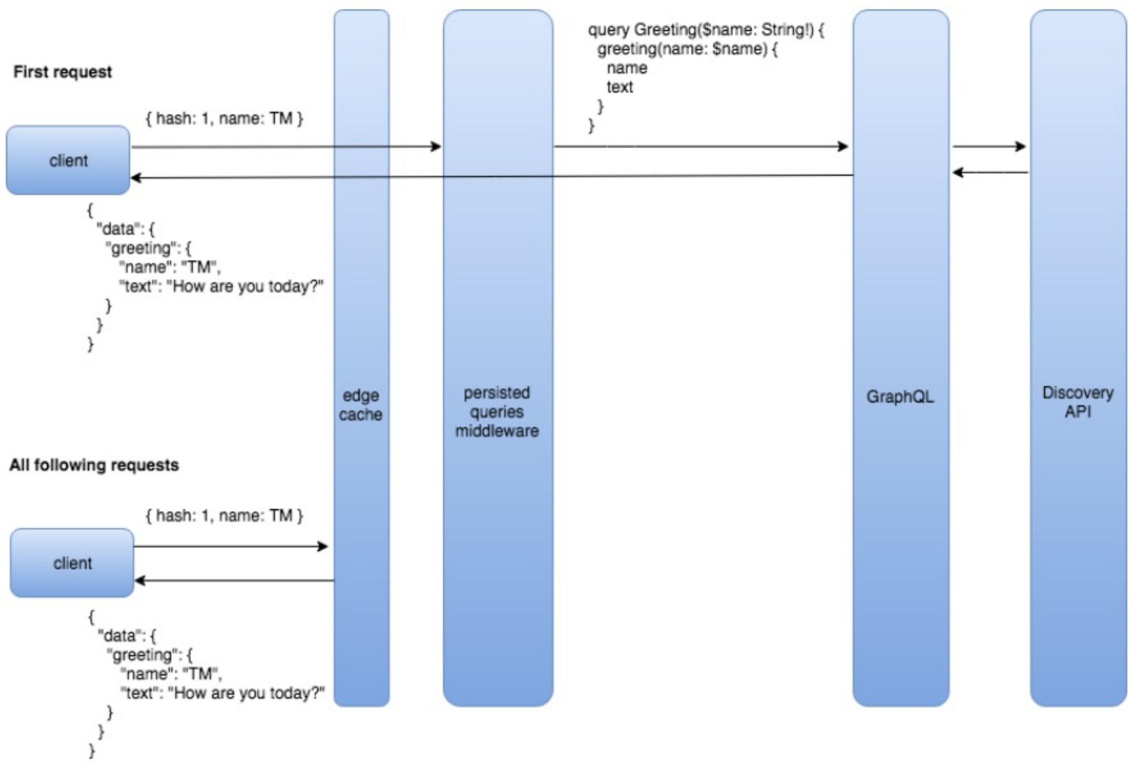
```
module.exports = {
 1: `query Greeting($name: String!) {
        greeting(name: $name) {
       name
       text
      }
    }`
}
```

- 如果hash存在，我们在获取到所有匹配的键：persistedQueries.js

```
const { omit } = require('ramda')
const queryMap = require('./extracted_queries.js')
const persistedQueries = (req, res, next) => {
  const { hash = '' } = req.query

  if (!hash) return next()
  const query = queryMap[hash]

  if (!query) {
    res.status(400).json({ error: [{}] })
    return next(new Error('Invalid query hash'))
  }
  req.query = {
    query,
    variables: omit(['hash'], req.query)
  }
  next()
}
module.exports = persistedQueries
```

在这个函数中，我们解析了query中的hash，如果不存在，返回到下一个组件中，如果存在，获取对应的query。。。

请求过程图解



动态graphql persist query

上面我们通过建立一个graphql服务器使用中间件来匹配一个查询的文件，我们知道在客户端和服务端保持静态文件的同步是一个不小的消耗，为了解决这个问题，我们通过引入动态graphql persist query, 我们可以创建一个react客户端，动态的生成hash值，然后把它传送给服务端，我们不仅更新graphql服务器来保持查询，而且还通过在内存中存储hash值对下次的查询。

example

- create-react-app函数创建一个react 客户端，redis-cli 内存数据库，然后再服务端安装redis客户端

```
npm install -g create-react-app redis-cli
```

- 服务端代码

```
const express = require('express')
const bodyParser = require('body-parser')
const { graphqlExpress } = require('apollo-server-express')
const { makeExecutableSchema } = require('graphql-tools')
const playground = require('graphql-playground-middleware-express').default
const persistedQueries = require('./persistedQueries')
const typeDefs = require('./schemas')
const resolvers = require('./resolvers')
const cors = require('cors')
const port = 4000
const app = express()
const schema = makeExecutableSchema({ typeDefs, resolvers })
app.use(
  '/graphql',
  cors(),
  bodyParser.json(),
  persistedQueries,
  graphqlExpress({ schema })
)
app.use(
  '/playground',
  playground({ endpointUrl: '/graphql' })
)
app.listen(port, () => console.log(`listening on port: ${port}`))
```

这里的cors 是redis的驱动包

- schema定义类型

```
const Greeting = `
  type Greeting {
    name: String
    age: Int
    profession: String
    text: String
  }
`
const Query = `
  type Query {
    greeting(name: String): Greeting
  }
`
module.exports = [Greeting, Query]
```

- resolver定义

```
const resolvers = {
  Query: {
    greeting: (_, { name }) => ({
      name,
      age: 99,
      profession: 'Software Engineer',
      text: 'how are you today?'
    })
  }
}

module.exports = resolvers
```

- 接下来可以通过react来创建query的hash

```
import gql from 'graphql-tag'

const getGreeting = gql`
  query Greeting($name: String) {
      greeting(name: $name) {
      name
      text
    }
  }
`

const extendGreeting = gql`
  query Greeting($name: String) {
      greeting(name: $name) {
      name
      age
      profession
      text
    }
  }
`

export {
  getGreeting,extendGreeting
}
```

如上代码中，我们添加了两个query文件，现在我们需要部分组件来调取我们的主页和两个额外的query

```
import React from 'react'

const Home = () =>
  <h1>This is the homepage</h1>

export default Home
```

- src/components/home.js

```
import React from 'react'

const Home = () =>
  <h1>This is the homepage</h1>
```

```
export default Home
```

- src/components/pageOne.js

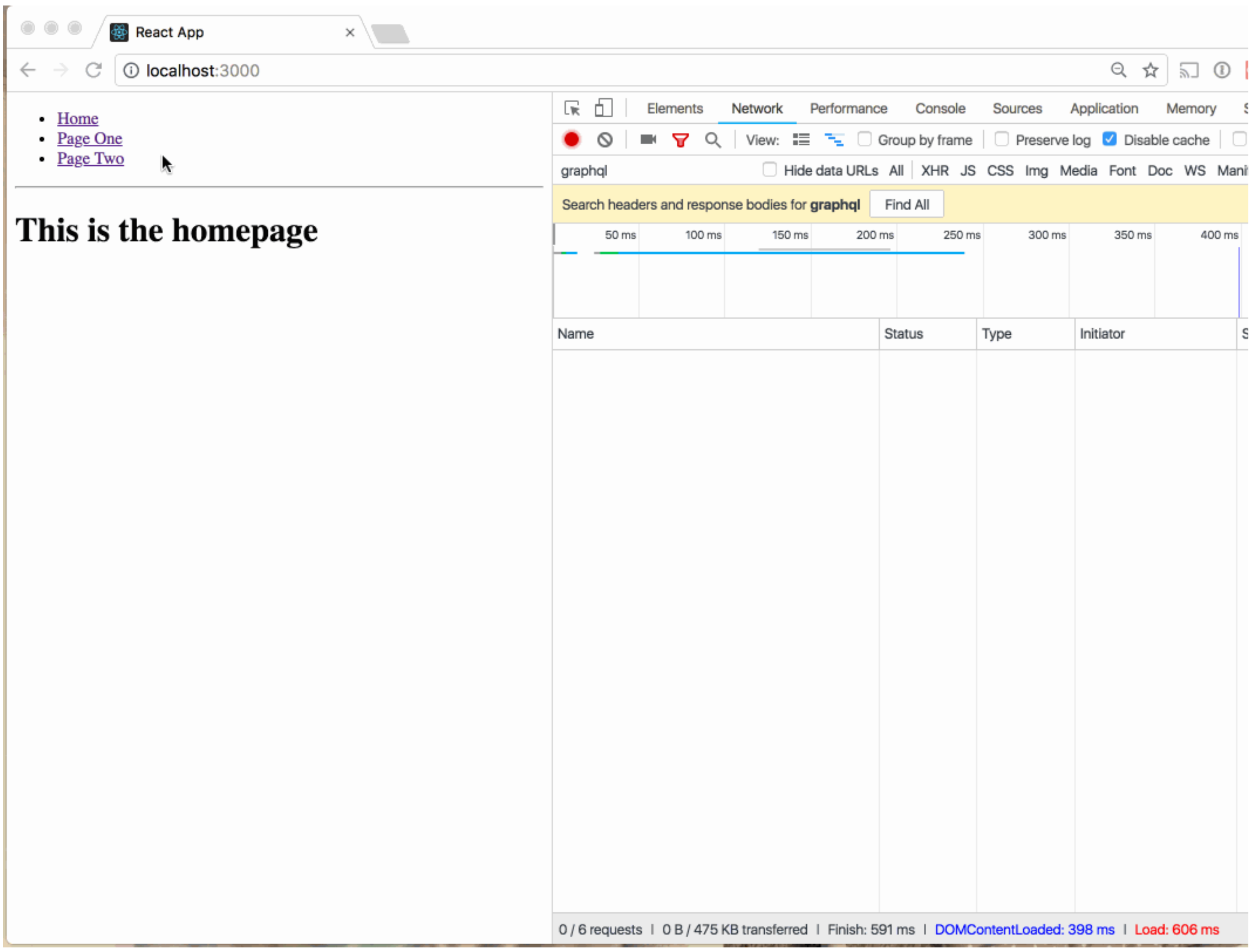
```
import React from 'react'
import { Query } from "react-apollo"
import { getGreeting } from '../queries'

const showGreeting = ({ error, loading, data: { greeting } }) => {
  if (loading) return 'Loading...'
  if (error) return `Error! ${error.message}`

  return <p>{greeting.name}, {greeting.text}</p>
}

const PageOne = () =>
  <Query query={getGreeting} variables={{ name: 'Cadence' }}>
    {showGreeting}
  </Query>

export default PageOne
```

我们的主页很简单的返回了一个string格式，pageOne和pageTwo组件通过query组件来发送request给graphql服务端，如果请求处于pending状态，我们显示文本loading，如果发生错误，我们显示错误信息，如果运行良好，我们返回需要的数据。

- persistQuery.js

```
const { parse, validate } = require('graphql')
const { makeExecutableSchema } = require('graphql-tools')
const Redis = require('ioredis')
const typeDefs = require('./schemas')

const client = new Redis()

const PQ = 'persisted-queries'

const schema = makeExecutableSchema({
  typeDefs,
})

const sendQuery = (req, next, data) => {
  req.query = data
  next()
}

const sendErrors = (res, next, msg, errors) => {
  res.json({ errors })
  return next(new Error(msg))
}

const handleQuery = (req, res, next, data) => results => {
  const {
    query,
    operationName,
    extensions: { persistedQuery: { sha256Hash } },
  } = data

  if (results != null)
    return sendQuery(req, next, { ...data, query: results })

  if (!query)
    return sendErrors(res, next, `Query hash not found: ${sha256Hash}`, [
      { message: 'PersistedQueryNotFound' },
    ])

  const errors = validate(schema, parse(query))

  if (errors.length)
    return sendErrors(
      res,
      next,
      `Invalid query provided with hash: ${query}`,
      errors
    )

  if (client.status !== 'ready')
    return sendQuery(req, next, data)

  const cacheKey = `${PQ}:${operationName}:${sha256Hash}`

  client
    .set(cacheKey, query)
    .then(() => sendQuery(req, next, data))
    .catch(err =>
      sendErrors(res, next, `Redis set pq error: ${err}`, [err]))
}

const method = req => {
  const isPostRequest = req.method === 'POST'
  const data = isPostRequest ? 'body' : 'query'

  const {
    extensions = isPostRequest ? {} : '{}',
  } = req[data]

  return isPostRequest
    ? { ...req[data], extensions }
    : { ...req[data], extensions: JSON.parse(extensions) }
}

const persistedQueries = (req, res, next) => {
  const data = method(req)

  const {
    operationName,
    extensions: { persistedQuery: { sha256Hash } = {} },
  } = data

  if (!sha256Hash) return next()

  if (client.status !== 'ready')
    return handleQuery(req, res, next, data)()

  const cacheKey = `${PQ}:${operationName}:${sha256Hash}`

  client
    .get(cacheKey)
    .then(handleQuery(req, res, next, data))
    .catch(err =>
      sendErrors(res, next, `Redis get pq error: ${err}`, [err]))
}

module.exports = persistedQueries

const method = req => {
  const isPostRequest = req.method === 'POST'
  const data = isPostRequest ? 'body' : 'query'

  const {
    extensions = isPostRequest ? {} : '{}',
  } = req[data]

  return isPostRequest
    ? { ...req[data], extensions }
    : { ...req[data], extensions: JSON.parse(extensions) }
}
```

效果展示



- Home
- Page One
- Page Two

This is the homepage

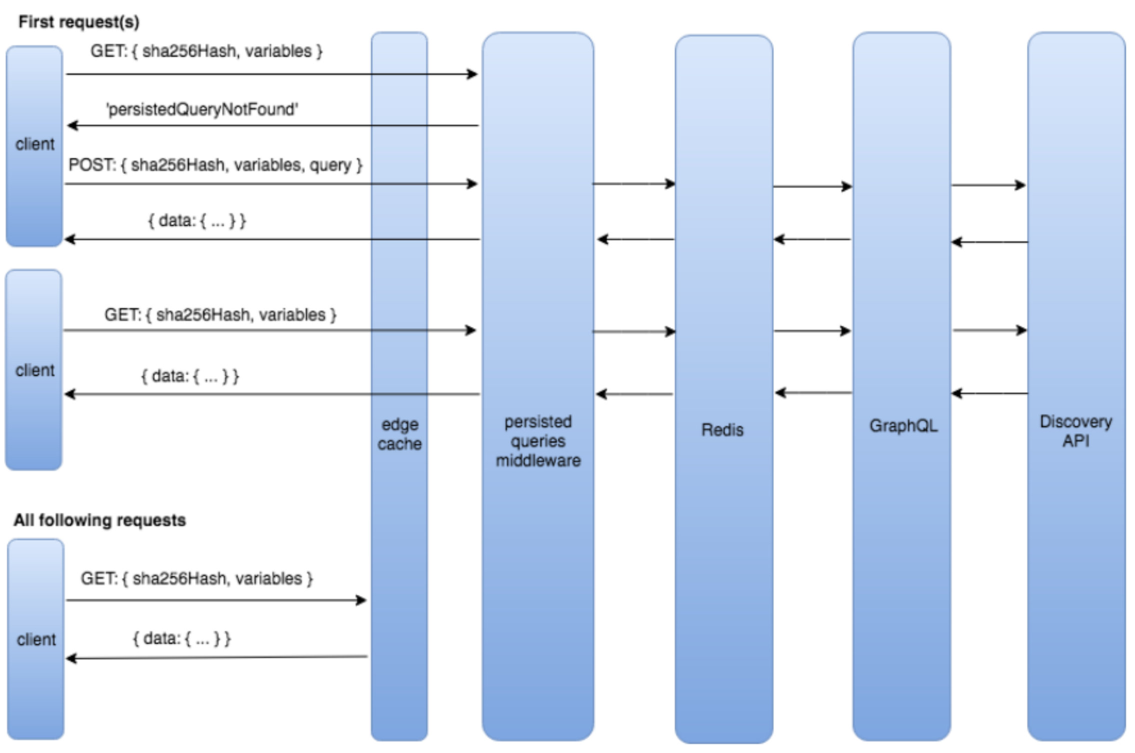
graphql

Search headers and response bodies for **graphql**

☑ Disable cache

0 / 6 requests | 0 B / 475 KB transferred | Finish: 591 ms | DOMContentLoaded: 398 ms | Load: 606 ms

请求过程图解



Each successful GET request can be cached on the edge by our CDN