GraphQL Party | 杭州

邓若奇

国内某大型电商前端开发专家

《基于 SPA 架构的 GraphQL 工程实践》

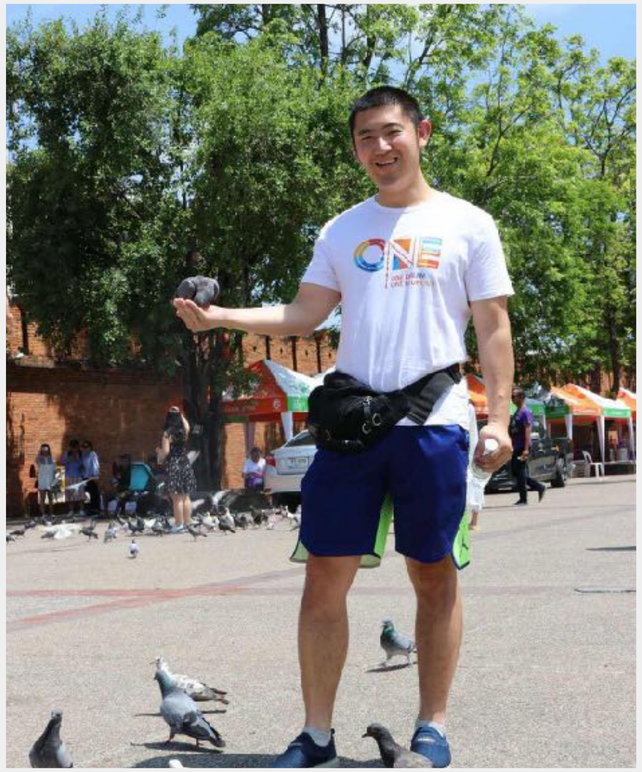
宋小菜 SONGXIAOCAI.COM x CODING CLOUD DEVELOPMENT

基于 SPA 架构的  工程实践

邓若奇

阿里巴巴 - CBU体验技术

SELF INTRODUCTION



邓若奇

5年前端/ nodes 开发，4年 .NET 开发。
阿里巴巴 - CBU体验技术 - 前端技术专家。
B2B 前端工程体系基建。
集团 nodejs 中间件客户端维护者及 DevOps。

Github: https://github.com/luckydrq
Email: ruoqi.drq@alibaba-inc.com
知乎: @邓若奇
Twitter: @luckydrq

CONTENTS

GraphQL 的哲学

GraphQL 的哲学

Schema Definition Language

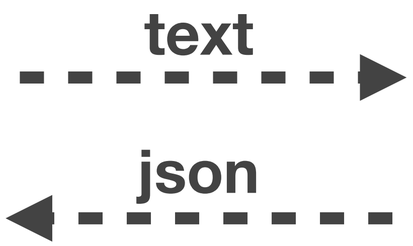
```
type User {
  id: ID!
  name: String
  blogs: [Blog]
  comments: [Comment]
}
```

```
type Blog {
  id: ID!
  content: String
  author: User!
  comments: [Comment]
}
```

```
type Comment {
  id: ID!
  content: String
  author: User!
  belongTo: Blog!
}
```

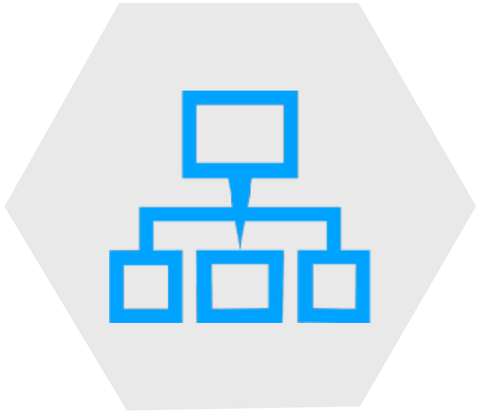
GraphQL 的哲学

```
{
  blog(id: 1) {
    content
    author {
      name
    }
    comments {
      content
      author {
        name
      }
    }
  }
}
```




```
{
  "content": "this is a blog about GraphQL",
  "author": {
    "name": "Bob"
  },
  "comments": [{
    "content": "Awesome!",
    "author": {
      "name": "Alice"
    }
  }, {
    "content": "Thank you!",
    "author": {
      "name": "Bob"
    }
  }]
}
```

GraphQL 的哲学



模型定义

按需查询

关系描述

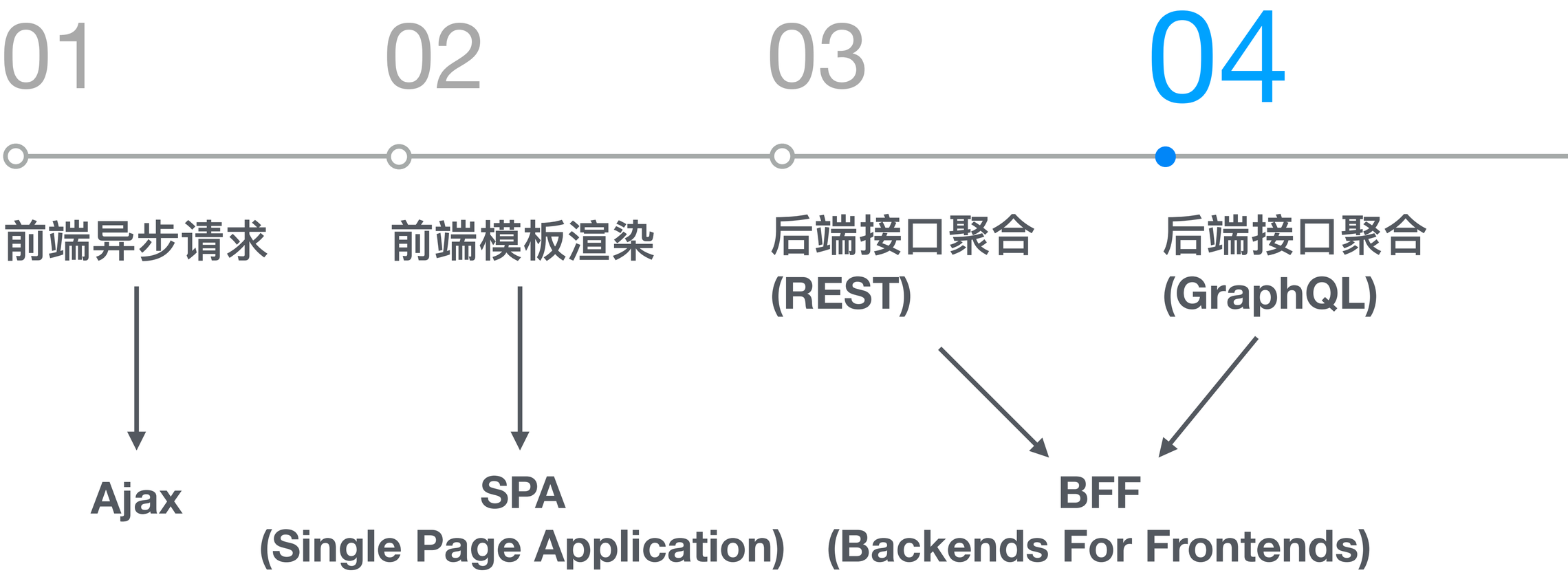
GraphQL 的哲学

Application Data Graph

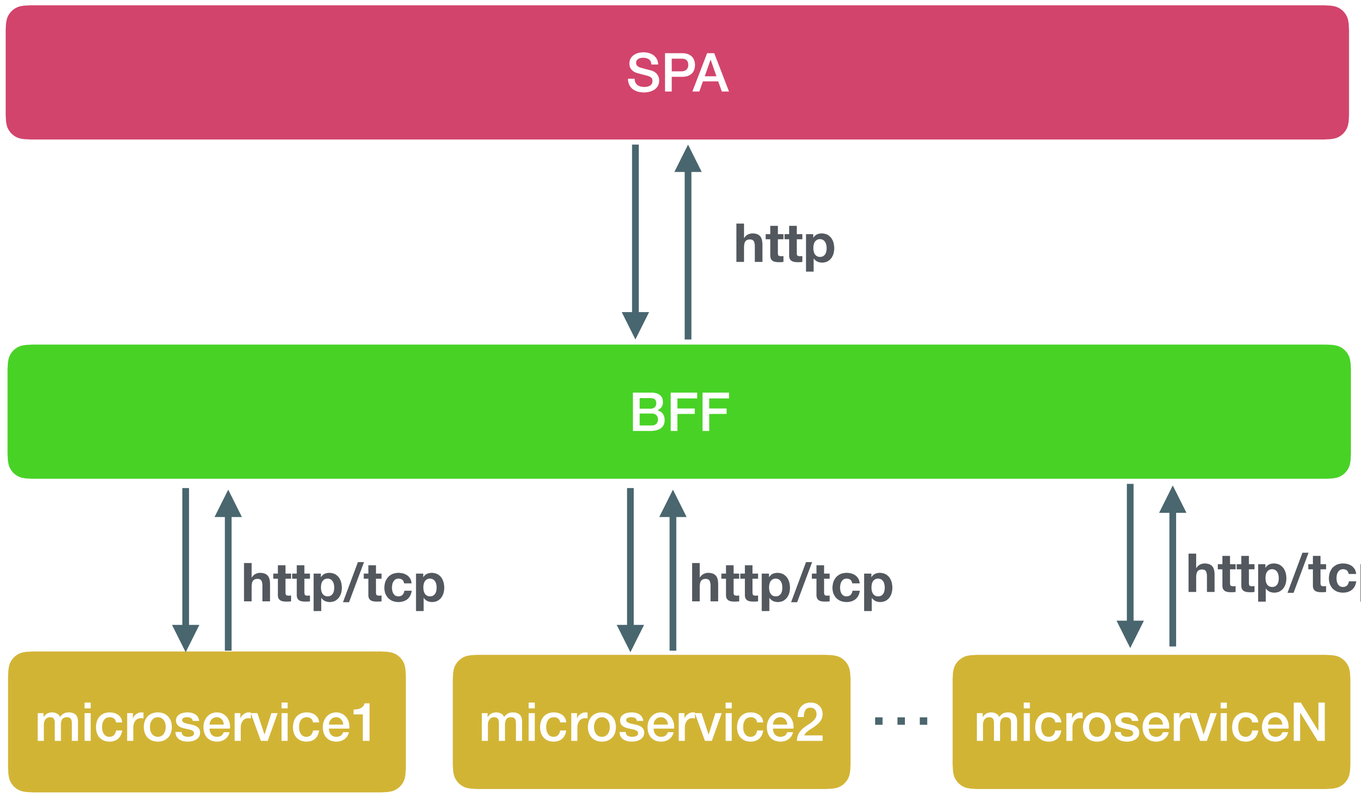
架构设计与技术选型

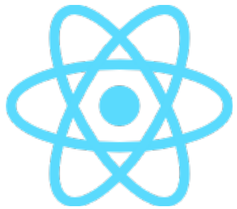
从前端视角看前后端分离

架构设计与技术选型

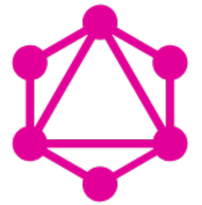

01
02
03
04
前端异步请求
前端模板渲染
后端接口聚合
(REST)
后端接口聚合
(GraphQL)
Ajax
SPA
(Single Page Application)
BFF
(Backends For Frontends)

架构设计与技术选型


SPA
http
BFF
http/tcp
http/tcp
http/tcp
microservice1
microservice2
···
microserviceN




egg



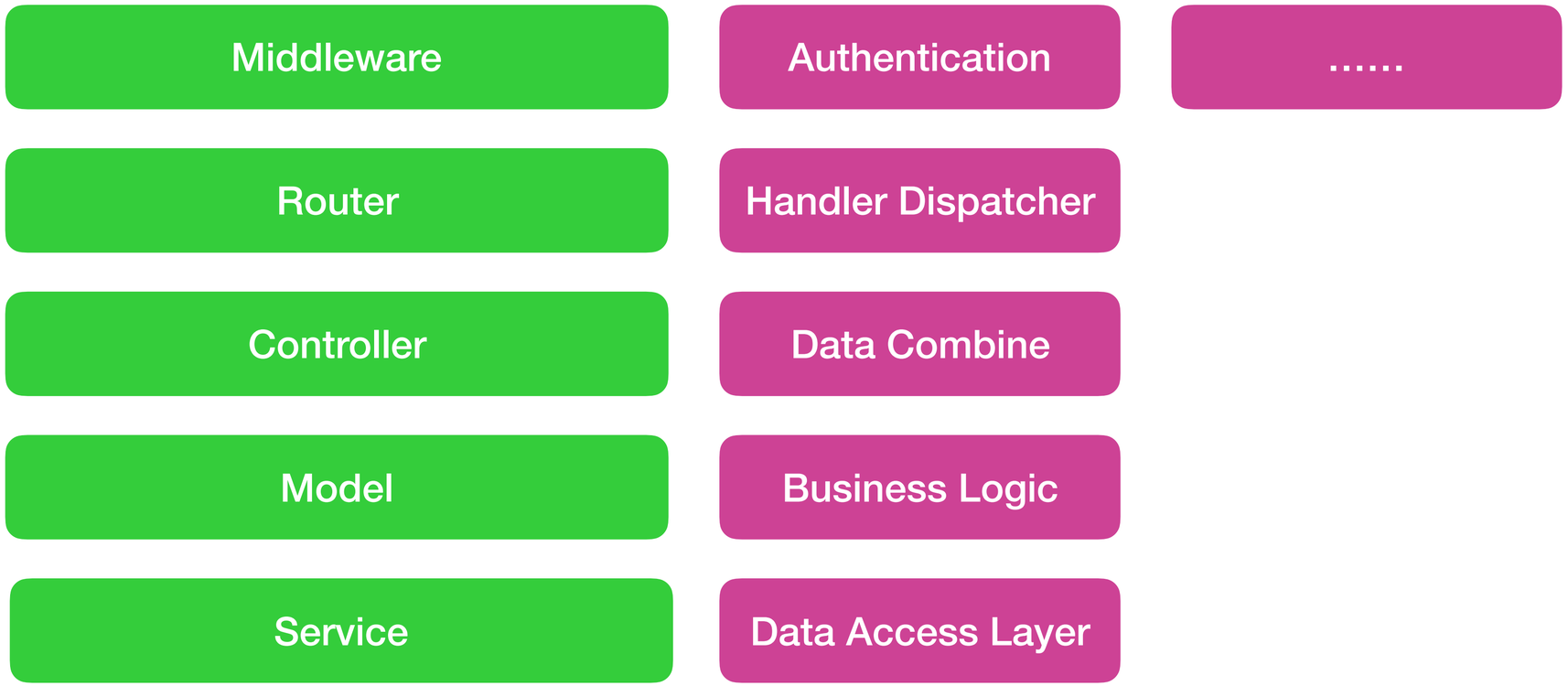

Java


node
JS

如何设计 BFF

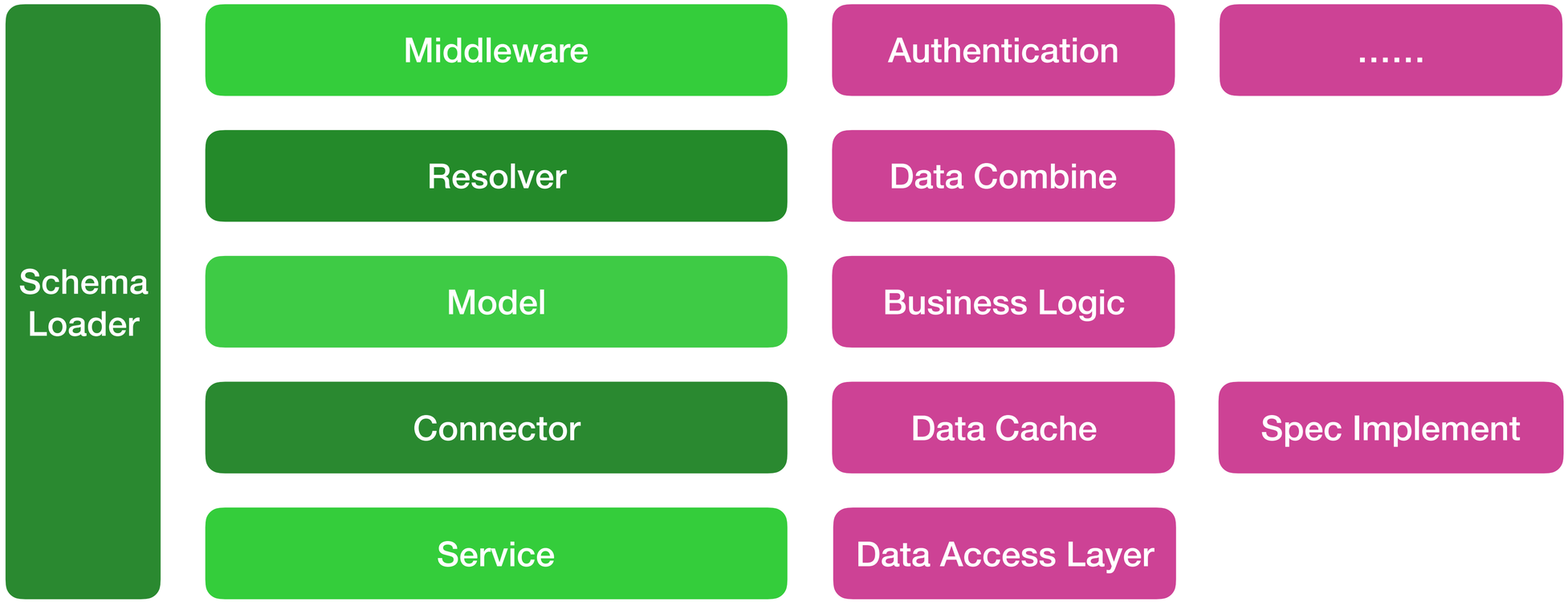
如何设计 BFF

基于 REST 的分层设计



如何设计 BFF

基于GraphQL的分层设计



如何设计 BFF

1、如何构建 Schema?

如何设计 BFF

```
import {
  GraphQLSchema,
  GraphQLObjectType,
  GraphQLString,
} from 'graphql';

var schema = new GraphQLSchema({
  query: new GraphQLObjectType({
    name: 'RootQueryType',
    fields: {
      hello: {
        type: GraphQLString,
        resolve() {
          return 'world';
        },
      },
    },
  }),
});
```

如何设计 BFF

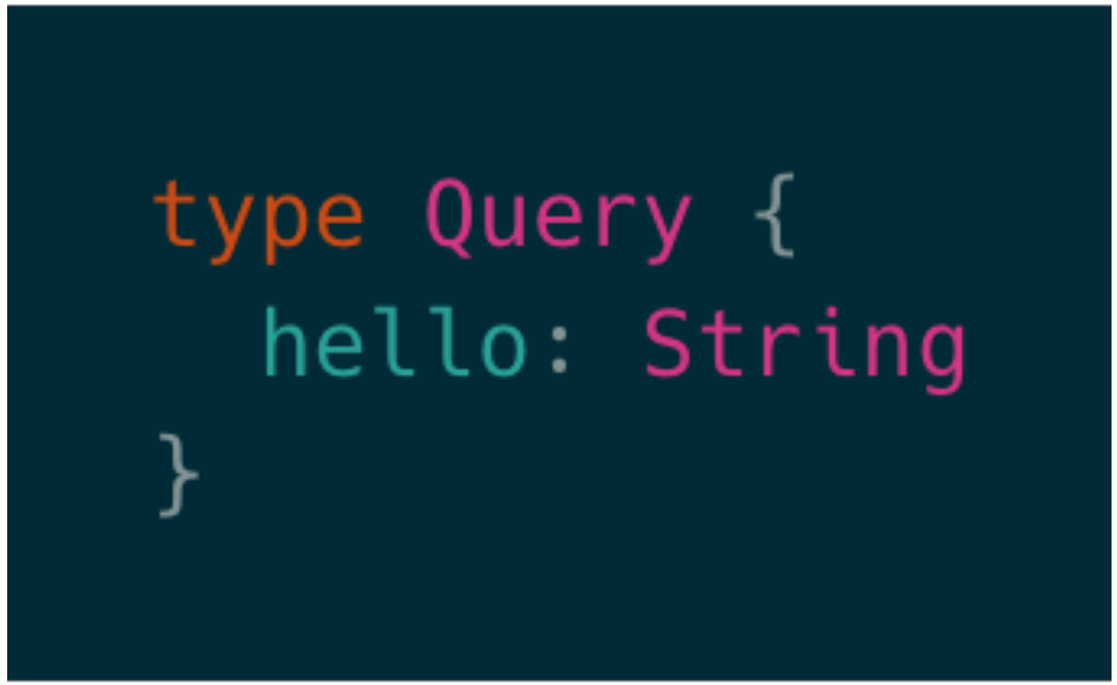
语言无关

设计先行

如何设计 BFF

SDL First Philosophy

如何设计 BFF



schema.graphql

resolver.js

如何设计 BFF

```
module.exports = app => {
  const schema = [ readSchema() ];
  const resolvers = require('./resolver')(app);

  app.graphqlSchema = makeExecutableSchema({
    typeDefs: schema,
    resolvers,
  });
};
```

2、鉴权与授权

如何设计 BFF

Authentication

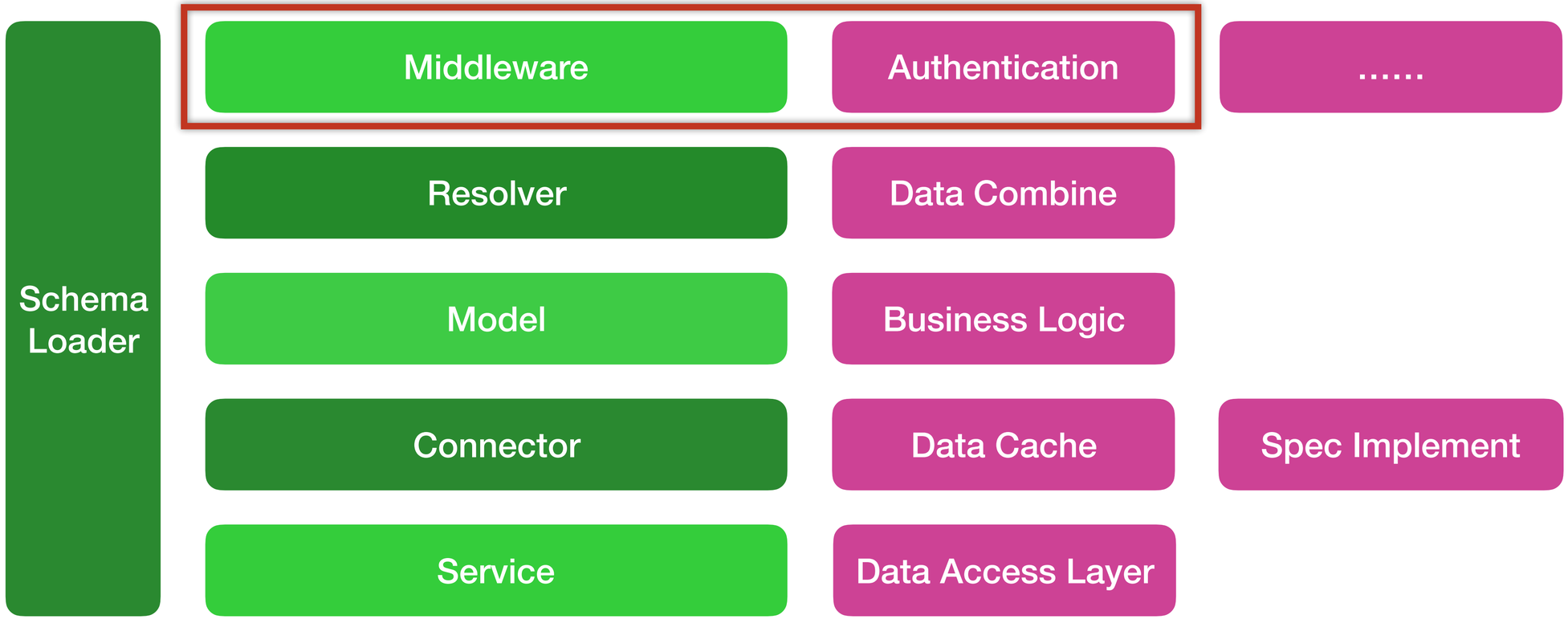
Authorization

如何设计 BFF

鉴权：通用，粗粒度

如何设计 BFF

基于 GraphQL 的分层设计



如何设计 BFF

授权：定制，细粒度

如何设计 BFF

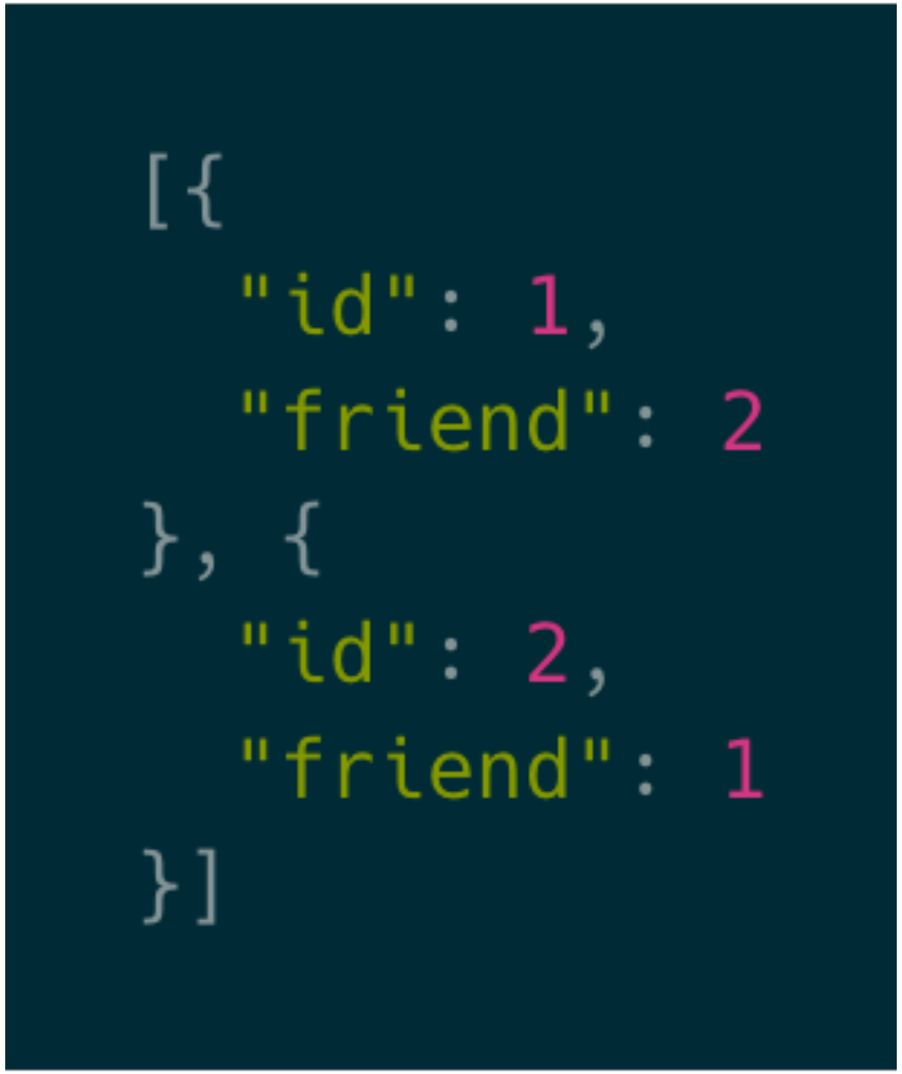
```
query {
  user(name: "小明") {
    salary
  }
}
```

如何设计 BFF

```
module.exports = {
  Query: {
    User: {
      salary({ salary }, { userId }, ctx) {
        const { User } = ctx.model;
        if (User.isSelf(userId)) {
          return salary;
        }
        return null;
      },
    },
  },
};
```

3、缓存设计

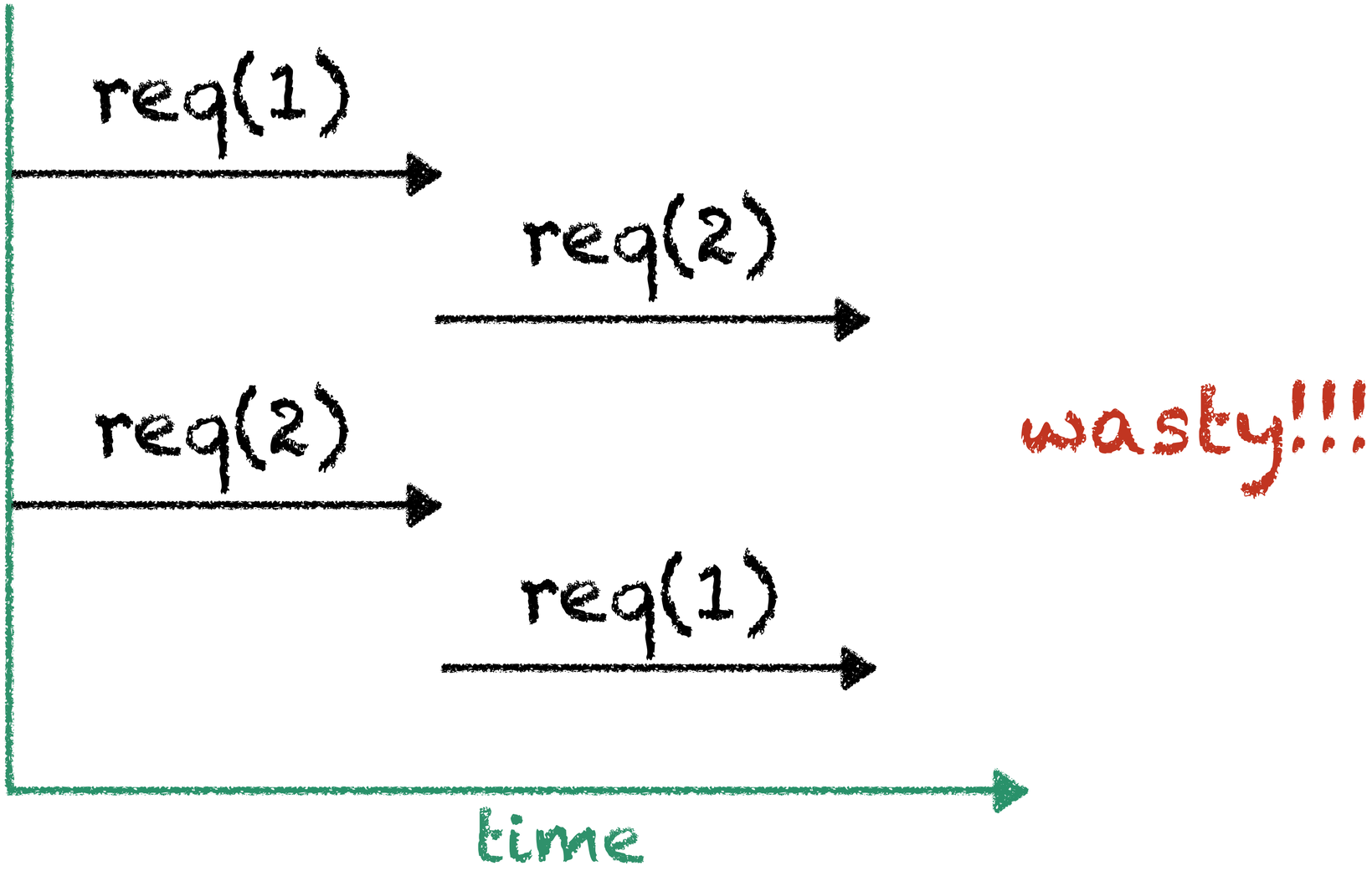
如何设计 BFF



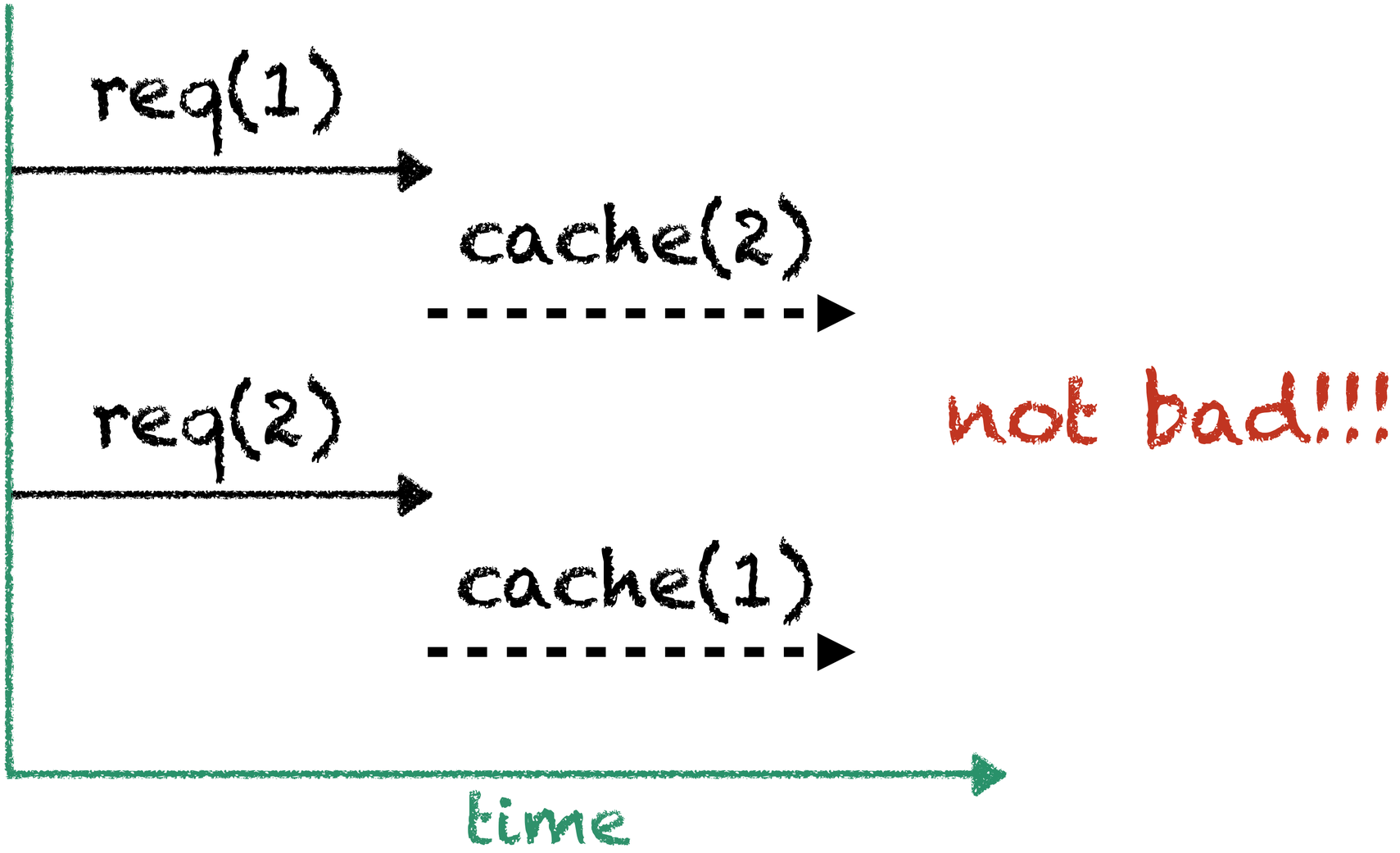
如何设计 BFF

```
userLoader.load(1)
  .then(user => userLoader.load(user.friend))
  .then(user => console.log(user.id)); // 2

userLoader.load(2)
  .then(user => userLoader.load(user.friend))
  .then(user => console.log(user.id)); // 1
```

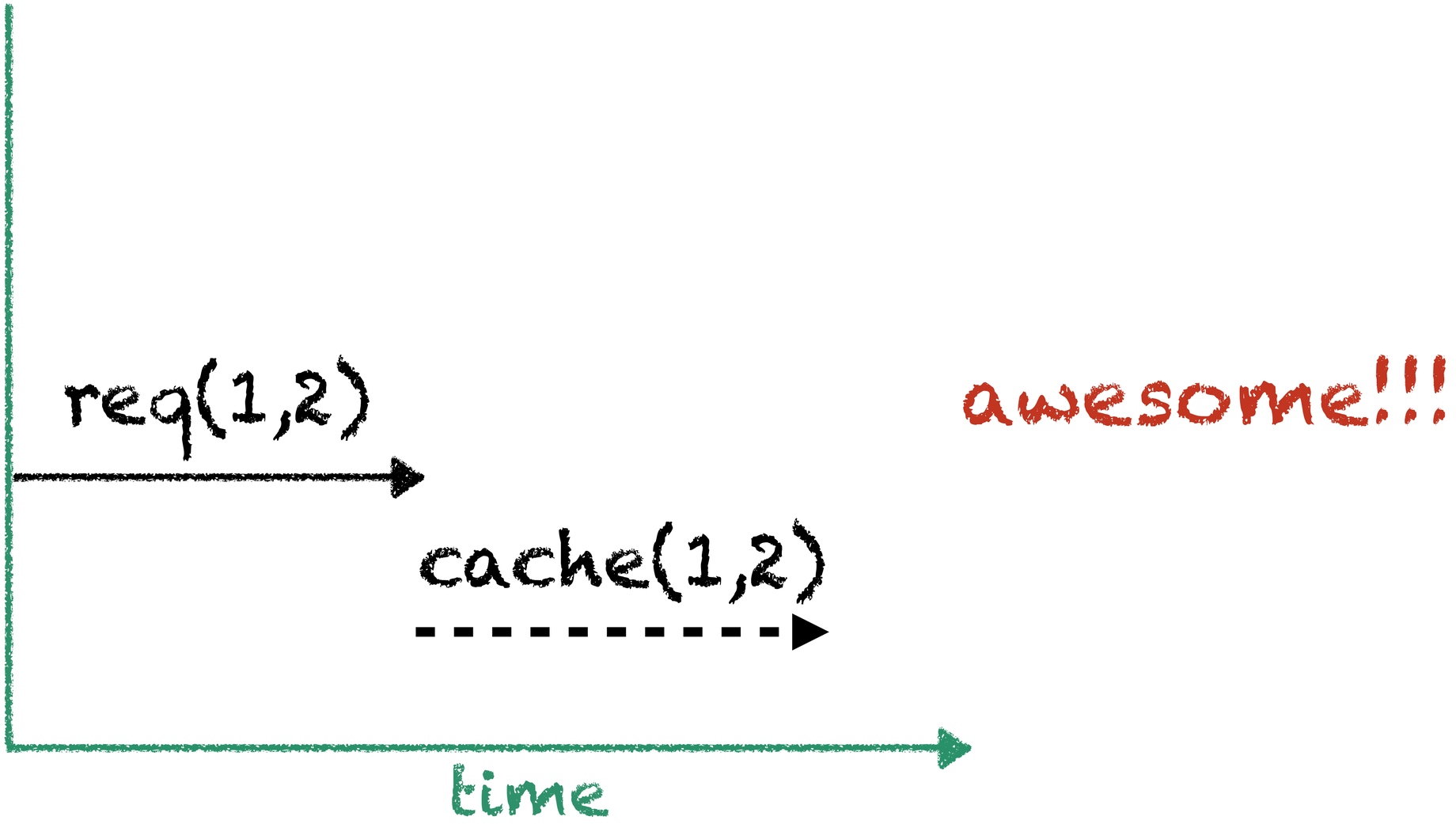
如何设计 BFF



如何设计 BFF



如何设计 BFF



如何设计 BFF

我们需要:

使用缓存 (caching)

请求队列 (queueing)

批量处理 (batching)

如何设计 BFF

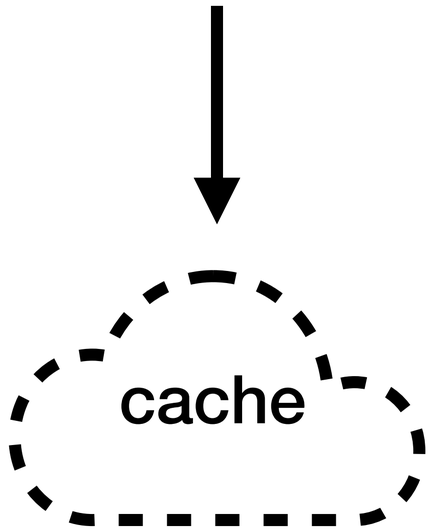
DataLoader

DataLoader is a generic utility to be used as part of your application's data fetching layer to provide a simplified and consistent API over various remote data sources such as databases or web services via batching and caching.

build passing coverage 100%

如何设计 BFF

new DataLoader(**keys** => {})



如何设计 BFF

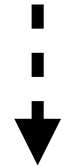
```
const DataLoader = require('dataloader');
const userLoader = new DataLoader(batchGetUsers);
const db = require('./db');

function batchGetUsers(ids) {
  return db.query('SELECT * FROM ?? WHERE id IN (?)', 'user', ids.join(','));
}
```

如何设计 BFF

userLoader.load(1)

userLoader.load(2)

userLoader.load(3)



userLoader.load(**[1,2,3]**)

SELECT * FROM **USER** WHERE **ID IN (1,2,3)**

如何设计 BFF

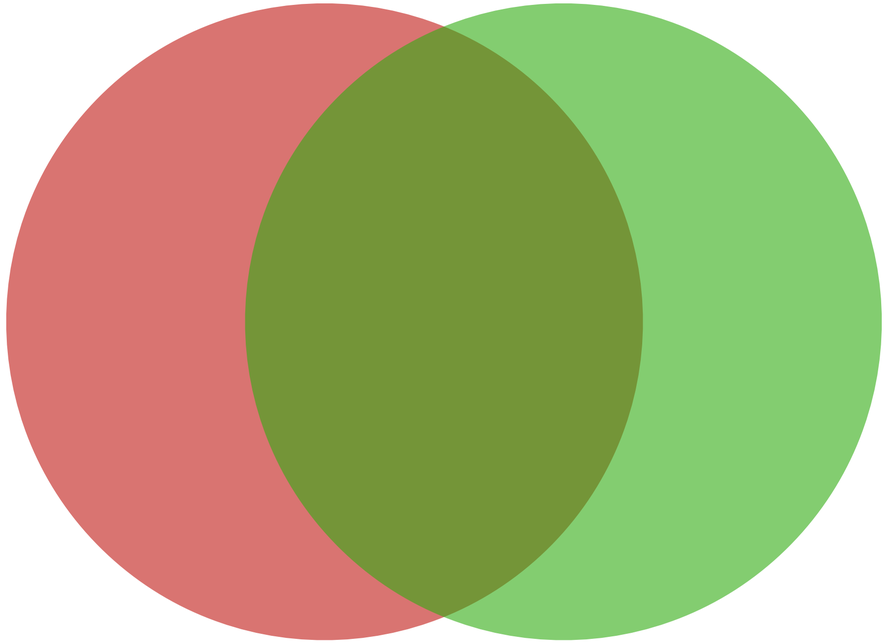
使用关系型数据库略复杂……

如何设计 BFF

```
const idLoader = new DataLoader(fetchByIds);
const mobileLoader = new DataLoader(fetchByMobiles);

idLoader.load(1)
mobileLoader.load('12345678');

function fetchByIds(ids) {
  return db.Users.getByIds(ids);
}

function fetchByMobiles(mobiles) {
  return db.Users.getByMobiles(mobiles);
}
```

如何设计 BFF

缓存利用率不高！



如何设计 BFF

rdb-dataloader

npm v1.0.1 | build passing | codecov unknown | dependencies up to date | vulnerabilities 0 | downloads 21/month

This module targets at relational database such as MySQL, SQL Server. Heavily inspired by Facebook DataLoader.

如何设计 BFF

```
const DataLoader = require('rdb-dataloader');
const uniqueKeyMap = new Map();
uniqueKeyMap.set('name', fetchByNames);  // UK
uniqueKeyMap.set('email', fetchByEmails);  // UK

const loader = new DataLoader(fetchByIds, { uniqueKeyMap });
loader.load('luckydrq', 'name')
  .then(record => {
    loader.load(record.id); // use cache if id matched
    done();
  }).catch(done);
```

如何设计 BFF

缓存记录的全部字段，数据量控制
应该由你的分页逻辑来关心

如何设计 BFF

思考：请求级缓存 VS 中心化缓存

前后端如何协作

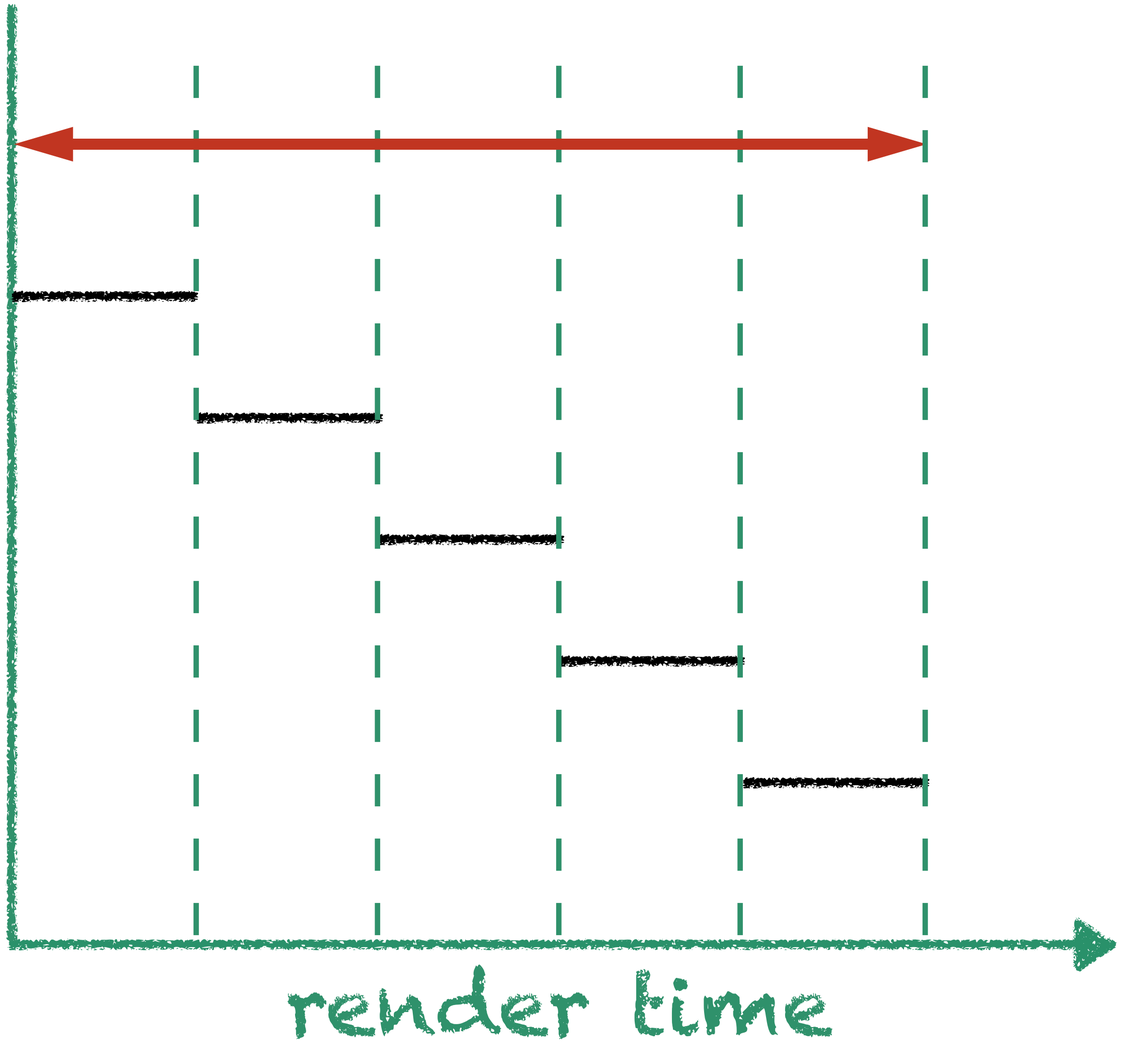
前后端如何协作

Thinking in Relay

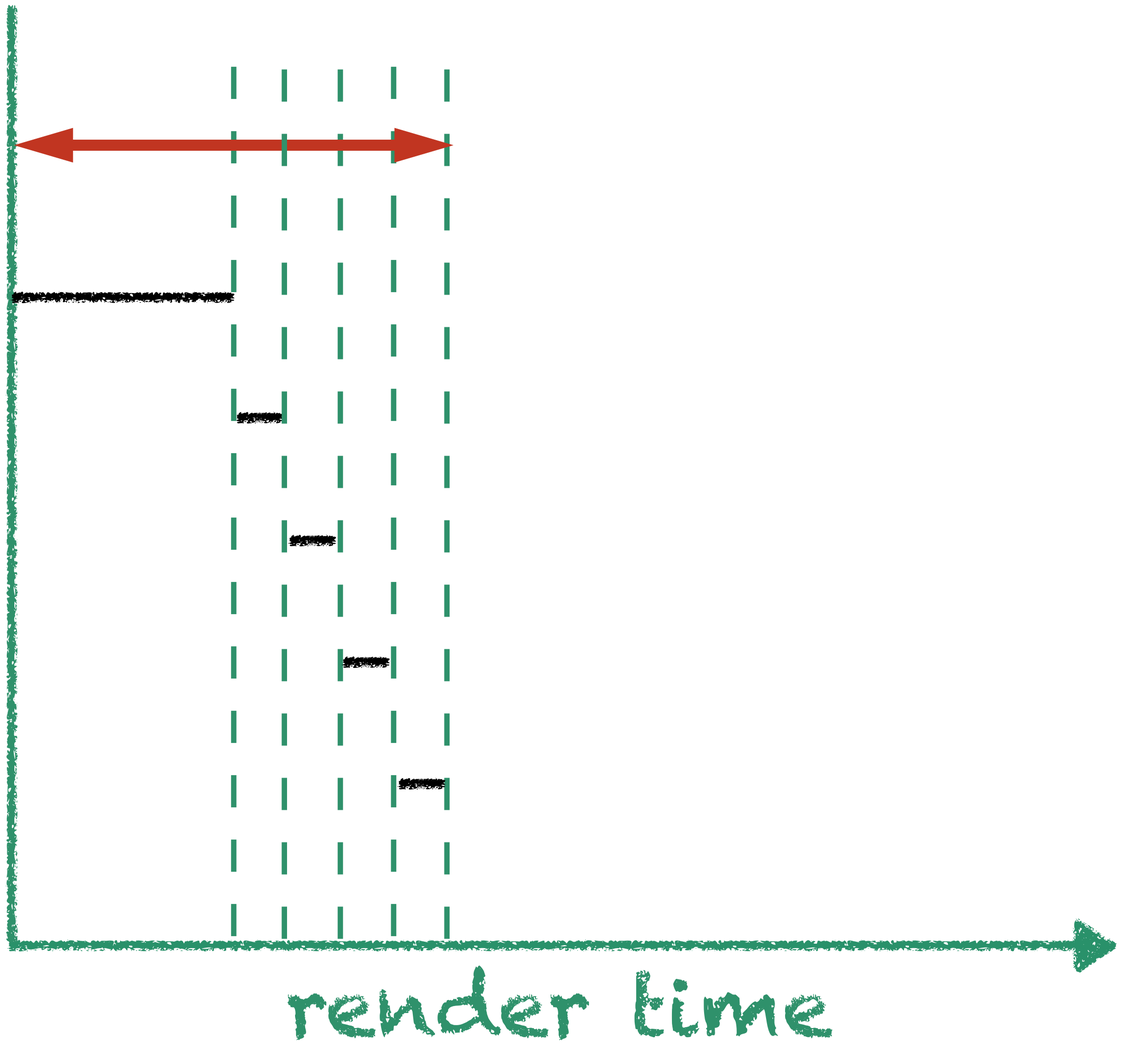
前后端如何协作

```
class Component extends React.Component {
  state = { data: null }

  componentDidMount() {
    fetch('./api.json')
      .then(data => this.setState({ data }))
      .catch(err => this.handleError(err));
  }

  render() {
    const { data } = this.state;
    return data ? <ChildComponent data={data} /> : null
  }
}
```

前后端如何协作



前后端如何协作



前后端如何协作

```
query totalQuery {
  user {
    id
    name
    mobile
    friend {
      id
      name
    }
    orders {
      id
      name
    }
  }
  orders {
    id
    name
    price
    creator {
      id
      name
    }
  }
}
```

前后端如何协作

```
import Order from './Order';
import { createFragmentContainer, graphql } from 'react-relay';

export default createFragmentContainer(Order, graphql`
  fragment Order_viewer on User {
    order {
      id
      name
      amount
    }
  }
`);
```

前后端如何协作

```
import { graphql } from 'react-relay';

export default AppQuery = graphql`
  query AppQuery {
    viewer {
      id
      name
      ...Order_viewer
    }
  }
`;
```

前后端如何协作

Cache a graph

前后端如何协作

```
Map {
  // Blog
  1: Map {
    content: 'this is a blog',
    author: Link(2),
  },
  // User
  2: Map {
    name: 'luckydrq',
  },
  // Comment
  3: Map {
    content: 'this is a comment',
    author: Link(2),
    belong: Link(1)
  }
}
```

View Consistency

前后端如何协作

```
Map {
  // Blog
  1: Map {
    content: 'this is a blog',
    author: Link(2),
  },
  // User
  2: Map {
    name: 'luckydrq',
  },
  // Comment
  3: Map {
    content: 'this is a comment',
    author: Link(2),
    belong: Link(1)
  }
}
```

前后端如何协作

Global Identifier

前后端如何协作

base64(type + ':' + id)

前后端如何协作

VXNlcjo2NDg0MA==	User
QnVpbGQ6MjM0MTk=	Build
QmluZGluZzo2	Binding
QnVpbGQ6MjM0MDk=	Build
QnVpbGQ6MjM0MDQ=	Build

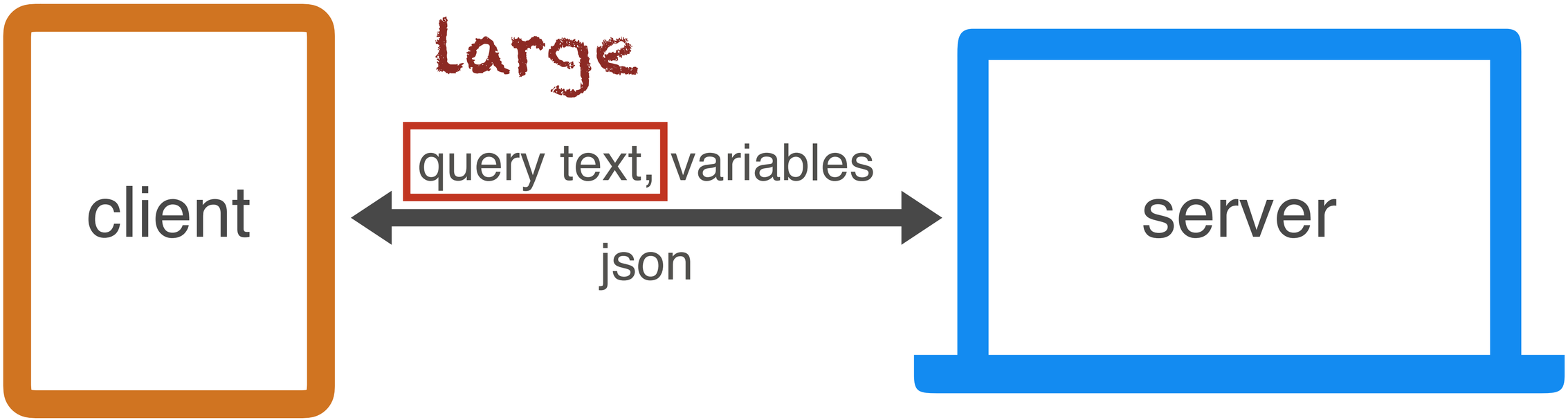
前后端如何协作

```
const { toGlobalId, fromGlobalId } = require('graphql-relay');

module.exports = app => {
  class BaseModelClass extends app.BaseContextClass {
    fromGlobalId(globalId) {
      if (typeof globalId === 'string') {
        const parsed = fromGlobalId(globalId);
        return parseInt(parsed.id, 10);
      }
      return globalId;
    }

    toGlobalId(id) {
      const type = this.constructor.name;
      return toGlobalId(type, id);
    }
  }
  return BaseModelClass;
};
```

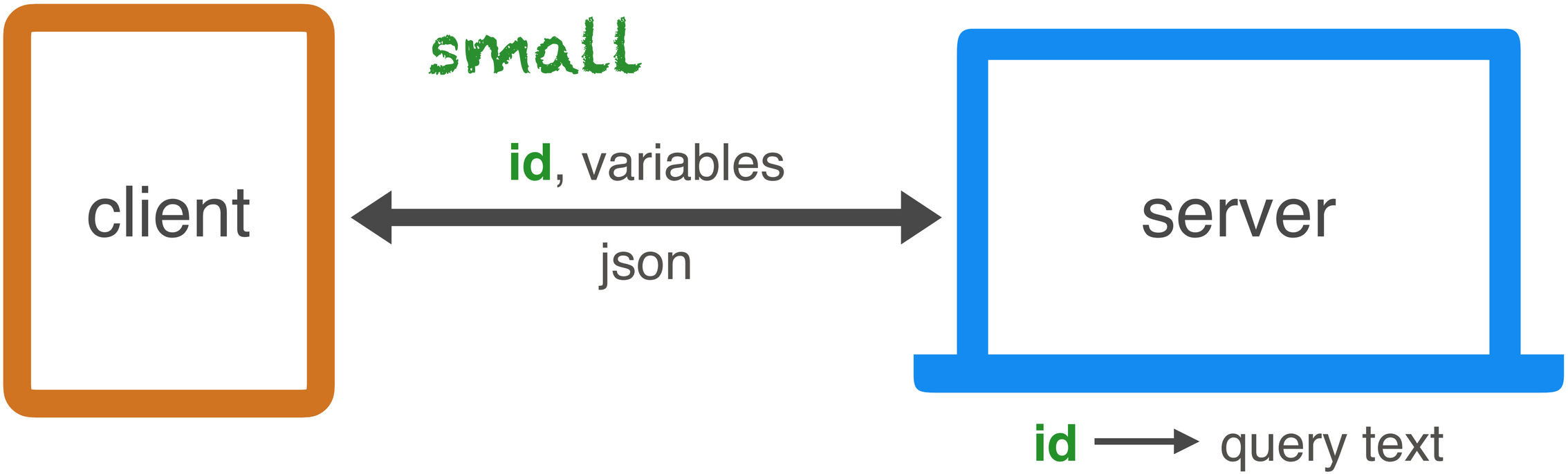
前后端如何协作

Persisted Query

前后端如何协作



前后端如何协作



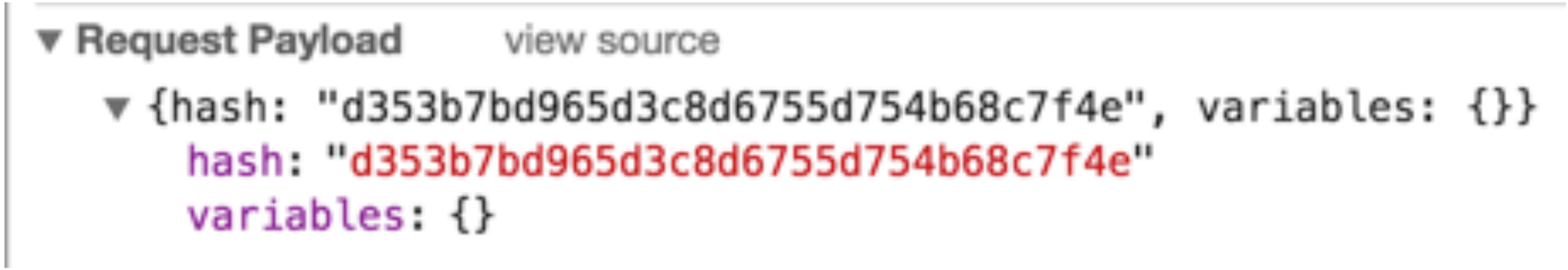
前后端如何协作

AOT Compile

前后端如何协作

```
const node/*: ConcreteRequest*/ = (function() {
  return {
    text: 'query HomeQuery {\n user \n{ id\n}',
    // xxxx
  }
})();
(node/*: any*/).hash = 'd353b7bd965d3c8d6755d754b68c7f4e';
module.exports = node;
```

前后端如何协作


Request Payload
view source
{hash: "d353b7bd965d3c8d6755d754b68c7f4e", variables: {}}
hash: "d353b7bd965d3c8d6755d754b68c7f4e"
variables: {}

需要解决的问题

需要解决的问题

Dos Attack

需要解决的问题

```
query {
  user(id: 1) {
    friend {
      friend {
        friend {
          friend {
            friend {
              friend {
                friend {
                  friend {
                    friend {
                      friend {
                        name
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
```

需要解决的问题

Size Limiting 

Query Whitelisting 

Depth Limiting 

需要解决的问题

```
const graphqlHTTP = require('koa-graphql');
const depthLimit = require('graphql-depth-limit');

module.exports = (options = {}, app) => {
  options.schema = app.graphqlSchema;
  options.validationRules = [ depthLimit(10) ];
  const mw = graphqlHTTP(options);

  return async function(ctx, next) {
    if (ctx.path === '/graphql') {
      const { query, hash } = ctx.request.body;
      // 有hash, 查询query text并设置
      if (!query && hash) {
        const q = app.queries[hash];
        if (q) {
          ctx.request.body.query = q.text;
        }
      }

      return await mw(ctx);
    }
    await next();
```

需要解决的问题

Rate Limiting

需要解决的问题

/graphql 

query/mutation 

需要解决的问题

```
type Mutation {
  addComment: Comment @rateLimit(limit: 20, window: 60000)
}
```

需要解决的问题

实现复杂但值得一试

TODO

Thank you !

Q&A

GraphQL Party | 杭州

宋小菜 SONGXIAOCAI.COM x CODING CLOUD DEVELOPMENT