S项目小结

业务

2022.4.25 入职，主要负责 B端【商品中心】和【红人营销】前端迭代和维护。

在商品中心，完成了整体能力建设：

1. 基础能力：围绕商品底层结构/核心发布及上下架流程，接入麦哲伦商品中台，实现对商品数据的统一管理，为 C端商城 提供必要的数据来源：
   a. 完成商品信息往 skc 层级下放，更精细管理商品数据字段；
   b. 销售库存实现全站共享+虚仓，保证库存信息正确，提升了售卖机会，减少超卖
   c. 接入标签库、摄影套图、板房商品尺码等，解藕常变数据，方便运营配置；
2. 业务支持：支持快速扩展国家、店铺渠道；支持不同模式商品运营；快速扩展商品品类；便于业务快速扩张：
   a. 支持主商品到站点商品运营，支持独立站商品和 Shopify、TTS、TTF等不同渠道商品运营，实现一品发多站的高效商品管理目标；
   b. 针对OBM新业务模式，协同供应商管理系统搭建商品品牌库，并打通C端展示链路，实现S的OBM商品品牌规范管理及应用；
   c. 新开欧洲7个国家支持独立站；支持TTF商品侧沙特开国；
3. 交互体验：优化交互体验；增加批量操作，提升运营效率：
   a. PLM系统上线后，打通了PLM到商品的推送链路，商品主数据直接读取PLM，减少信息录入成本，业务粗估由1人日创建25款提升到1人日可创建70款，效率提升150%+；
   b. 支持商品各环节打回PLM，减少线下沟通成本；协同业务对齐和摄影分工，支持商品创建活动图灵活管理，减少非必要流程；
   c. 完成批量上下架、excel批量导入导出、批量审核、PLM产品批量上架排期、商品更新前后便捷对比、批量更新多个站点信息、长宽高打通WMS减少输入项、标题配置规则自动生成并引入算法自动生成、UED专项迭代30+优化点等多项批量及效率提升功能；

数据：目前已建设一级类目19，叶子类目476，属性200+，属性值2000+，商标近100，已完成TTS/自建站/Shopify3个渠道、17个站点的商品统一管理，已支持业务发布主商品7700+，站点商品7.7w。

在支持沙特上线项目，月上架数2000+，月新增品类100+。

技术

S项目B端由很多子应用构成，目前已有十来个，对于项目的规范和基建支持很有需要。

参与技术项目：

1. BFF 建设：
   a. 调研 gulu NodeJS 框架，为 guluS 插件搭建做准备
   RPC 使用介绍 、@byted-service/rpc 浅析 、服务发现 Consul 、链路信息 Metainfo 、Gulu 问题排查
   b. 搭建 guluS 插件脚手架，统一代码规范、类型支持、构建发包。降低开发负担，提高插件保障度。
   guluS-脚手架
   c. 开发维护 guluS 透传插件，减少繁琐的 rpc 接口透传处理，为 wms 多端系统快速适配自定义路由前缀，支持透传接口耗时记录。
   https://bnpm.bytedance.net/package/@gulu-s/plugin-rpc-forward
   d. 开发维护 response 处理插件，规范 BFF 返回结构。
   https://bnpm.bytedance.net/package/@gulu-s/plugin-response

e. 参与 plm 以图搜图开发；完成 mysql 接入、设计、监控告警等流程；完成 BFF 定时任务建设。

2. arcoS 组件库建设：

   a. 完善组件脚手架，统一代码规范、单测覆盖、构建发包，规范化版本管理，接入 ci/cd 自动化发包。降低开发负担，提高开发效率。
   [WIP]构建发布自动化 、单测接入、存量组件统一规范

3. 研发规范建设：

   a. 开发 eslint 插件，覆盖团队前端项目、组件库项目，统一代码规范。
   Eslint 使用介绍及插件开发指南 、https://bnpm.bytedance.net/package/@s-fe/eslint-plugin

   b. 调研 mock 方案，提供开发效率。
   Mock 指北

   c. 接入飞书机器人，服务后端通过 goofy stack 接入 gulu NodeJS，方便后续搭建自动化通知，自动化报表能力。
   S-FE robot、goofy stack

   d. 调研并推动团队接入火车发布
   S 供应链前端接入火车发布

代码示例

前端很多自动化的工作都能基于目录结构

1. 在前端项目中，webpack 提供了 api `requie.context` ，能够在编译阶段获取目录信息。对于B端系统，一个简单的使用场景，是通过目录结构，自动化生成路由配置，菜单栏配置，以及对路由和菜单权限的绑定。以下是获取目录结构的一个小例子

```ts
import {
  RouteProps,
  useLocation,
  matchPath,
} from '@jupiter/plugin-runtime/router';

export interface IRoute extends RouteProps {
  [key: string]: any;
  path: string; // 路由地址
  redirect?: string; // 重定向
  routes?: IRoute[]; // 子路由
  layout?: React.ReactNode; // 路由组件包裹层
  hideSideMenu?: boolean; // 是否隐藏 SideMenuBar
  showProductMenu?: boolean; // 是否展示产品菜单
  componentProps?: Record<string, any>; // 组件 props
}

/**
 * 判断是否是路由配置
 * @param val
 * @returns boolean
 */
const isRoute = (val: any): val is IRoute => typeof val?.path === 'string';

/**
 * 过滤正确的路由配置
 * @param val
 * @param res
 * @returns IRoute[]
 */
const filterRoute = (val: any, res: IRoute[] = []) => {
  if (isRoute(val)) {
```

```
      res.push(val);
    } else if (Array.isArray(val)) {
      val.forEach(itm => filterRoute(itm, res));
    }

    return res;
};

/**
 * 检索 @/router 目录下所有路由配置，即从所有 index.ts 文件里获取路由配置
 */
const genRoutes = () => {
  let res: IRoute[] = [];
  const r = require.context('./', true, /index\.ts$/);
  r.keys().forEach(key => {
    // 排除此文件
    if (key !== './index.ts') {
      const vals = Object.values(r(key));
      res = res.concat(filterRoute(vals));
    }
  });

  return res;
};

// 路由汇总
const allRoutes = genRoutes();
export default allRoutes;

/**
 * 通过 Location 获取当前 current route
 * @returns Iroute
 */
export const useGetRoute = () => {
  const { pathname } = useLocation();
  const tmp = [...allRoutes];
  const recur = (arr: IRoute[]) => {
    for (const itm of arr) {
      if (matchPath(pathname, itm)) {
        return itm;
      }

      // 拼接到数组后面，直至遍历所有 routes
      if (itm.routes) {
        arr.push(...itm.routes);
      }
    }

    return null;
  };
  return recur(tmp);
};

```

2. 在 NodeJS 环境，不需要担心同步阻塞代码的情况，有很多方式能够获取目录结构。这边的一个小场景，是通过获取 monorepo 中所有子模块包名，结合 `commitizen` ，对 git commit scope 做一个约束，用于 ci/cd 集成。

```
// get-packages.js
const path = require('path');
```

```
const globby = require('globby');
const loadJsonFile = require('load-json-file');

// 获取所有子模块 package.json
const getPackages = () => {
  const allPackageJson = globby.sync(
    `${path.resolve(__dirname, '../')}/components/*/package.json`,
  );
  return allPackageJson.map(loadJsonFile.sync);
};

const allPackages = getPackages();

const packageNames = allPackages.map(v => v.name.replace(/@arco-s\//, ''));

module.exports = {
  allPackages,
  packageNames,
};
```

前端组件做结构封装，hooks做逻辑封装

B段系统列表页是一个主要的页面结构，对搜索区和表格的逻辑封装在 hooks 里，减少重复编码，更好维护

```
import { useState, useEffect, useCallback, useRef } from 'react';
import {
  PaginationProps,
  Message,
  FormInstance,
  Notification,
} from '@arco-design/web-react';
import { useRequest, useStateRealtime } from '@byted/hooks';
import { QueryFilterProps } from '@s_op/components/src/queryFilter/index.modules';
import { DEFAULT_SIZE_OPTIONS, DEFAULT_PAGE_SIZE } from '@/constants';
import { PaginationInfo } from '@/api/common/common';
import { trimFormData } from '@/utils/form';
import { BaseResp } from '@/bff/product/namespaces/base';

interface IPage {
  pagination_info: {
    page_no: number;
    page_size: number;
  };
}

type IRowKey = string | number;

export interface ISearchList<Req, TableItem> {
  // rowKey
  rowKey: string;
  // 页面初始化，一般是请求预设数据接口
  init?: () => Promise<void>;
  // 处理请求参数，一般处理时间窗参数
  reqParamsFormat?: (params: Partial<Req>) => Req;
  // 搜索列表
  searchList: (params: Partial<Req & IPage>) => Promise<{
    paginationInfo?: PaginationInfo;
    list: TableItem[];
  } | null>;
  // downloadType?: any;
```

```
37      // downloadKey?: keyof Req; // 导出请求字段
38      ready?: boolean; // ready 实现按需触发，是否初始化就查询，默认是
39      selectCrossPage?: boolean; // 是否开启表格跨页选择
40      defaultReqData?: Partial<Req>;
41      exportRequest?: () => Promise<{ base_resp?: BaseResp }>;
42    }
43
44    /* eslint-disable max-statements */
45    const useSearchList = <
46      Req extends IObject = IObject,
47      TableItem extends IObject = IObject,
48    >({
49      rowKey,
50      init,
51      reqParamsFormat,
52      searchList,
53      // downloadType,
54      // downloadKey,
55      ready = true,
56      selectCrossPage = true,
57      defaultReqData = {},
58      exportRequest,
59    }: ISearchList<Req, TableItem>) => {
60      type IReq = Req & IPage;
61
62      // searchForm ref
63      // const formRef = useRef<FormInstance<Req>>(null);
64      const formRef = useRef<FormInstance>(null);
65      formRef.current?.setFields(defaultReqData);
66
67      // 翻页参数
68      const [pagination, setPagination] = useState<PaginationProps>({
69        sizeOptions: DEFAULT_SIZE_OPTIONS,
70        total: 0,
71        current: 1,
72        pageSize: DEFAULT_PAGE_SIZE,
73        size: 'small',
74        onChange: (pageNum, pageSize) => {
75          setPagination(cur => ({
76            ...cur,
77            current: pageNum,
78            pageSize,
79          }));
80        },
81      });
82
83      // 请求参数
84      const [reqData, setReqData, getReqData] = useStateRealtime<IReq>({
85        ...defaultReqData,
86        pagination_info: {
87          page_no: 1,
88          page_size: DEFAULT_PAGE_SIZE,
89        },
90      } as IReq);
91      // 列表
92      const [tableList, setTableList] = useState<TableItem[]>([]);
93      const [selectedRowKeys, setSelectedRowKeys] = useState<IRowKey[]>([]);
94      const [selectedRowDatas, setSelectedRowDatas] = useState<TableItem[]>([]);
95
96      const updateSelectedRowDatas = useCallback(
97        (oldDataKeys: IRowKey[], newDateKeys: IRowKey[] = []) => {
98          const oldDatas = selectedRowDatas.filter(v =>
```

```
        oldDataKeys.includes(v[rowKey]),
      );
      const newDatas = tableList.filter(v => newDateKeys.includes(v[rowKey]));
      setSelectedRowDatas([...oldDatas, ...newDatas]);
    },
    [selectedRowDatas, tableList, rowKey],
  );

  // 表格多选数据
  const handleRowKeysChange = useCallback(
    (keys: IRowKey[], datas: TableItem[]) => {
      if (selectCrossPage) {
        // 当前页所有 keys
        const curPageKeys: IRowKey[] = tableList.map(v => v[rowKey]) ?? [];
        // 移除当前页所有选中项
        if (keys.length === 0) {
          const newKeys = selectedRowKeys.filter(
            s => !curPageKeys.includes(s), // 移除当前页所有选中项
          );
          setSelectedRowKeys(newKeys);
          updateSelectedRowDatas(newKeys);
        } else {
          // 如果所选 keys 不在已选 selectedRowKeys 中，则为新页
          const isNewPage = !keys.some(k => selectedRowKeys.includes(k));
          if (isNewPage) {
            // 新页直接合并
            setSelectedRowKeys(oldKeys => [...oldKeys, ...keys]);
            setSelectedRowDatas(oldDatas => [...oldDatas, ...datas]);
          } else {
            // 当前页未选中项
            const notSelectedKeys = curPageKeys.filter(c => !keys.includes(c));
            // 删除未选中项
            let newKeys = selectedRowKeys.filter(
              k => !notSelectedKeys.includes(k),
            );

            // 新的选中项
            const newSelectedKeys = keys.filter(
              k => !selectedRowKeys.includes(k),
            );
            newKeys = [...newKeys, ...newSelectedKeys];
            setSelectedRowKeys(newKeys);
            updateSelectedRowDatas(newKeys, newSelectedKeys);
          }
        }
      } else {
        setSelectedRowKeys(keys);
        setSelectedRowDatas(datas);
      }
    },
    [
      tableList,
      selectedRowKeys,
      rowKey,
      selectCrossPage,
      updateSelectedRowDatas,
    ],
  );

  // 查询，同时保存请求参数reqData
  const search = useCallback(
    async (params: IReq) => {
```

```
161         // console.log('search params', params);
162         setReqData(params);
163         const res = await searchList(params);
164         if (res) {
165           const { list, paginationInfo } = res;
166           setTableList(list ?? []);
167           setPagination(state => ({
168             ...state,
169             total: paginationInfo?.total_count,
170           }));
171         }
172       },
173       [searchList, setReqData],
174     );
175     const {
176       run: runSearch,
177       loading,
178       error,
179     } = useRequest(search, {
180       ready, // ready 实现按需触发
181     });
182
183     useEffect(() => {
184       if (error) {
185         Notification.error({ content: error.message });
186       }
187     }, [error]);
188
189     // searchForm 搜索按钮触发查询
190     const request = useCallback(
191       async (data: Partial<Req>) => {
192         const params = reqParamsFormat?.(data) ?? data;
193         const newState: IReq = {
194           ...reqData,
195           ...params,
196           pagination_info: {
197             page_no: 1, // 重置页码
198             page_size: pagination.pageSize,
199           },
200         };
201
202         setPagination(state => ({
203           ...state,
204           current: 1, // 重置页码
205         }));
206
207         setReqData(newState);
208         await runSearch(newState);
209         handleRowKeysChange([], []); // 清空表格选中项
210       },
211       [
212         pagination,
213         reqParamsFormat,
214         handleRowKeysChange,
215         reqData,
216         setReqData,
217         runSearch,
218       ],
219     );
220
221     // searchForm 回车触发查询
222     const handleSearch = useCallback(async () => {
```

```
    const fromData = formRef.current?.getFieldsValue();
    await request(fromData ?? {});
  }, []);

  // 页面初始化，请求预设接口
  const _init = useCallback(async () => {
    await Promise.all([runSearch(reqData), init?.()]);
  }, [reqData, init, runSearch]);
  const {
    run: runInit,
    loading: pageLoading,
    error: initError,
  } = useRequest(_init);
  useEffect(() => {
    runInit();
  }, []);

  useEffect(() => {
    if (initError) {
      Notification.error({ content: initError.message });
    }
  }, [initError]);

  // 翻页触发查询
  useEffect(() => {
    const { pagination_info } = getReqData();
    const { page_no, page_size } = pagination_info;
    const { current, pageSize } = pagination;

    if (current !== page_no || pageSize !== page_size) {
      const newState = {
        ...getReqData(),
        pagination_info: {
          page_no: pagination.current!,
          page_size: pagination.pageSize!,
        },
      };

      runSearch(newState);
    }
  }, [pagination, runSearch]);

  // 导出
  const exportList = useCallback(async () => {
    try {
      const { base_resp } = (await exportRequest?.()) ?? {};
      if (base_resp?.code === 0) {
        Message.success('已开始导出，请到「右上角-下载中心」查看进度');
        handleRowKeysChange([], []); // 清空表格选中项
        window?.Garfish.channel.emit('main-update-download-center');
      } else {
        throw new Error(base_resp?.message);
      }
    } catch (err: any) {
      console.error(err);
      Message.error(err?.message);
    }
  }, [handleRowKeysChange, exportRequest]);
  const { run: runExportList, loading: exportLoading } = useRequest(exportList);

  // 去空格
  const trimQeqData = useCallback(() => {
```

```
    if (formRef.current) {
      trimFormData(formRef.current);
    }
  }, []);

  return {
    pageLoading,
    tableLoading: loading,
    formRef,
    tableList,
    setTableList,
    selectedRowKeys,
    selectedRowDatas,
    pagination,
    request,
    handleSearch,
    handleRowKeysChange,
    exportList: runExportList,
    exportLoading,
    trimQeqData,
  };
};
/* eslint-enable */

type IReturnType = ReturnType<typeof useSearchList>;

export interface ISearchForm extends QueryFilterProps {
  formRef: IReturnType['formRef'];
  handleSearch: IReturnType['handleSearch'];
  trimQeqData: IReturnType['trimQeqData'];
}

export default useSearchList;
```

利用继承减少重复编码但保留个体特性（多态）

1. NodeJS 中对 mysql 的 crud 操作都是类似的，因此可以很好利用这一点

```
// base.ts
import { HttpContext } from '@gulu/application-http';
import {
  // Model,
  ModelStatic,
  CreationAttributes,
  CreateOptions,
  Attributes,
  UpdateOptions,
  FindAndCountOptions,
  FindOptions,
  DestroyOptions,
  UpsertOptions,
  CountOptions,
} from '@gulu/sequelize';
// 不使用这个类型会报错
import { Model } from '@byted-service/sequelize-creator/node_modules/sequelize-typescript';
import BaseService from '../base';
import { TimestampsModel } from '../../util/base/model';

export default abstract class Base<
```

```
  P extends IObject,
  C extends IObject = P,
  M extends Model = Model<P, C>
> extends BaseService {
  protected LOG_PREFIX = '[mysql]';
  private model: ModelStatic<M>;
  private timestamps: boolean;

  constructor(ctx: HttpContext, model: ModelStatic<M>) {
    super(ctx);

    this.model = model;
    this.timestamps = TimestampsModel.isPrototypeOf(model);
  }

  get repo() {
    return this.ctx.sequelize.getRepository(this.model);
  }

  async create(params: CreationAttributes<M>, options?: CreateOptions<Attributes<M>>) {
    const tmp = Object.create(null);
    if (this.timestamps) {
      const now = Date.now().valueOf();
      tmp.createTime = now;
      tmp.updateTime = now;
    }
    const res = await this.repo.create({ ...tmp, ...params }, options);

    return res?.dataValues as P;
  }

  async update(params: { [key in keyof Attributes<M>]?: Attributes<M>[key] }, options:
UpdateOptions<Attributes<M>>) {}

  async findOne(options?: FindOptions<Attributes<M>>) {}

  async findAll(options?: FindOptions<Attributes<M>>) {}

  async delete(options: DestroyOptions<Attributes<M>>) {}

  async deleteOneByID(value: number | string, key = 'id') {}

  async upsert(params: CreationAttributes<M>, options?: UpsertOptions<Attributes<M>>) {}

  async count(options?: Omit<CountOptions<Attributes<M>>, 'group'>) {}
}
```

```
// picture.ts
import { HttpContext } from '@gulu/application-http';
import { Op } from '@gulu/sequelize';
import PictureModel, { TAttributes, CAttributes } from '../../model/picture';
import Base from './base';

class Picture extends Base<TAttributes, CAttributes> {
  LOG_PREFIX = '[mysql/Picture]';

  constructor(ctx: HttpContext) {
    super(ctx, PictureModel);
```

```
    }

    async hasUri(uriList: string[]) {
      const count = await this.count({
        where: {
          uri: {
            [Op.in]: uriList,
          },
        },
      });
      return count > 0;
    }
  }

  export default Picture;

```

接口服务合理利用 Promise

1. 为接口提供最大返回时长，超过时长则返回已处理数据

```
  const sleep = (timeout: number) =>
    new Promise(resolve => {
      setTimeout(resolve, timeout);
    });

  type AnyFuncion = (...args: any[]) => any;
  class Sleep {
    callbacks: AnyFuncion[];
    timeout: number;

    constructor(timeout: number) {
      this.callbacks = [];
      this.timeout = timeout;

      this.run();
    }
    subscript(callback: AnyFuncion) {
      if (typeof callback === 'function') {
        this.callbacks.push(callback);
      }
    }
    notify() {
      this.callbacks.forEach(callback => {
        callback();
      });
    }
    async run() {
      await sleep(this.timeout);
      this.notify();
    }
  }

  const onTime = function <T>(promises: Promise<T>[], timeout: number): Promise<(T | undefined)[]> {
    return new Promise(resolve => {
      const result: (T | undefined)[] = Array(promises.length);

      promises.forEach(async (promise, index) => {
        const res = await promise;
        result[index] = res;
```

```
40
41          if (result.length >= promises.length) {
42            resolve(result);
43          }
44        });
45
46        new Sleep(timeout).subscript(() => {
47          resolve(result);
48        });
49      });
50    };
51
52    export default onTime;
```

2. 定时任务，对大批量操作实行分批处理

```
1   import { HttpApplication } from '@gulu/application-http';
2   import schedule from 'node-schedule';
3   import monent from 'moment';
4   import { DAY_TIMESTAMP } from '../util/constant';
5   import { Corpus } from '../extension/context';
6   import { TAttributes as TPictureCache } from '../model/picture_cache';
7   import { TAttributes as TUriRelation } from '../model/uri_relation';
8   import chunk from 'lodash/chunk';
9
10  type DBItem = TPictureCache & TUriRelation;
11
12  // 定时任务，凌晨2点刷新近7日图片查重信息
13  export const fetchPictureMatchInfoOn2am = async (app: HttpApplication) => {
14    const LOG_PREFIX = '[Schedule]';
15    const DAY_FORMAT = 'YYYY-MM-DD';
16    const SEVEN_DAYS_TIMESTAMP = 7 * DAY_TIMESTAMP;
17    const ctx = app.mockContext();
18    const {
19      logger,
20      service: { mysql, picture, redis },
21      sequelize,
22    } = ctx;
23
24    const rule = new schedule.RecurrenceRule();
25    rule.tz = 'PRC'; // 时区，默认为中国时区，后续如果多国家需要重新适配为用户所在地
26    rule.hour = 2;
27    rule.minute = 0;
28    // rule.second = 0;
29
30    schedule.scheduleJob(rule, async () => {
31      logger.info(`${LOG_PREFIX} - Batch fetch picture match info on 2:00am`);
32
33      const now = monent();
34      const redisKey = `${LOG_PREFIX}-${now.format(DAY_FORMAT)}`;
35      const unlocked = await redis.base.setLock(redisKey);
36      // 通过 redis 锁，避免 nodejs 多实例执行多次轮询
37      if (!unlocked) {
38        logger.info(`${LOG_PREFIX} - redis lock ${redisKey} existed, stop the cron job`);
39        return;
40      }
41
42      const [rows] = (await sequelize.query(
43        `SELECT * from picture_cache LEFT JOIN uri_relation ON picture_cache.source = uri_relation.uri
    AND picture_cache.create_time > ${
```

```
        now.valueOf() - SEVEN_DAYS_TIMESTAMP
      }`
    )) as [DBItem[], any];

    const unit = async ({ source, code, codeType, corpusName, extraInfo }: DBItem) => {
      let parseExtraInfo: Record<string, any> = {};
      try {
        parseExtraInfo = JSON.parse(extraInfo) ?? {};
      } catch (err) {
        parseExtraInfo = {};
      }
      await picture.getOneMatchInfo({
        uri: source,
        code,
        codeType,
        corpus: corpusName?.includes('hotselling') ? Corpus.HOTSELLING : Corpus.DEFAULT,
        isTest: corpusName?.includes('test'),
        url: parseExtraInfo.url,
      });
    };

    // 50条为一批串行执行，降低db负载
    await chunk(rows, 50).reduce(
      async (res, arr) =>
        res.then(async () => {
          await Promise.all(arr.map(unit));
        }),
      Promise.resolve()
    );
  });

  // 防止node宕机容错
  process.on('SIGINT', async () => {
    logger.error(`${LOG_PREFIX} - Gracefully shutdown`);
    await schedule.gracefulShutdown();
    process.exit(0);
  });
};

```

团队

- 参与 S 项目外包招聘，大概面试了50个人

候选人管理
人员面试
人员定级

待面试 已面试

候选人 职位类别 所在城市 操作日期: 2021/04/25- 2023/02/15 面试方式 面评结果 供应商 清空筛选项

候选人	职位类别	职位名称	所在城市	操作日期	面评结果	面试官	供应商
刘艳明	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-23	终止面试	吴一伟 已停用	纬创软件（武汉）有限公司
孙宇玲	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-11	淘汰	黄浩扬	博彦科技承德有限公司
刘佳颖	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-09	终止面试	黄浩扬	博彦科技承德有限公司
王校佳	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-09	终止面试	黄浩扬	博彦科技承德有限公司
刘妍	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-09	面试爽约	黄浩扬	博彦科技承德有限公司
陈家轩	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-05	终止面试	吴一伟 已停用	亿达信息技术有限公司
李聪聪	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-05	淘汰	黄浩扬	博彦科技承德有限公司
刘炜	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-08-04	未面试	黄浩扬	博彦科技承德有限公司
刘晓春	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-07-28	淘汰	黄浩扬	博彦科技承德有限公司
李玉娇	前端/后端开发工程师	前端/后端开发工程师-电商项目-...	北京	2022-07-27	终止面试	林佳枫 已停用	纬创软件（武汉）有限公司

共 52 条 10条/页 1 2 3 4 5 6 前往 1 页