> 包括初期对需求的分析，自己开发步骤， 开发遇到的问题，解决方案

初期

1. 看交互稿，看2遍大概看懂了，但是没具体看数据
2. 需求评审，pm测试策划前端参与，后端缺席
3. ui没完全出来，看部分figma，继续结合交互

开发步骤

1. 先是加载页ui+预加载功能实现+熟悉利用umi layouts这个配置路由的功能 并实现加载动画

```
useEffect(() => {
    (async () => {
      // 先预留时间加载首图再进行预加载，减少无背景图的情况下显示加载进度条
      await new Promise((resolve) => {
        setTimeout(() => {
          resolve(1);
        }, 1000);
      });
      const preloadImages: string[] = [];
      const context = require.context('@/assets/img/', true, /\.jpg|\.png$/);
      function importAll(r: __WebpackModuleApi.RequireContext) {
        // function importAll(r) {
        r.keys().forEach((key) => {
          const realUrl = r(key);
          if (!realUrl.startsWith('data:image')) {
            preloadImages.push(realUrl);
          }
        });
      }
      importAll(context);
      preloadInstance.loadManifest(preloadImages);
      const time = Date.now();
      preloadInstance.on('progress', (item: any) => {
        setProgress(Math.round(item.progress * 100));
      });
      preloadInstance.on('complete', () => {
        setTimeout(() => {
          readyCallback?.();
          // console.log('asdas');
        }, 1000);
        console.log('preload  time is: ', Date.now() - time);
      });
    })();
  }, [readyCallback]);
```

2. 路由设计 所有页面框架大致搭建
3. 绑定页各种状态实现，这次就比上次王牌竞速少耗时很多
4. 通用数据页抽象封装，重构了很多次
5. 数据接口调用的问题，这次后端比较晚才出接口，运营也是比较晚提供数据样式。还有账号数据的问题需要多方协调才有测试数据

6. 埋点文档，这次还需要传数据，所以要结合数据逻辑，埋的时候还发现文档的很多不细致和错误的地方和dm进行确认

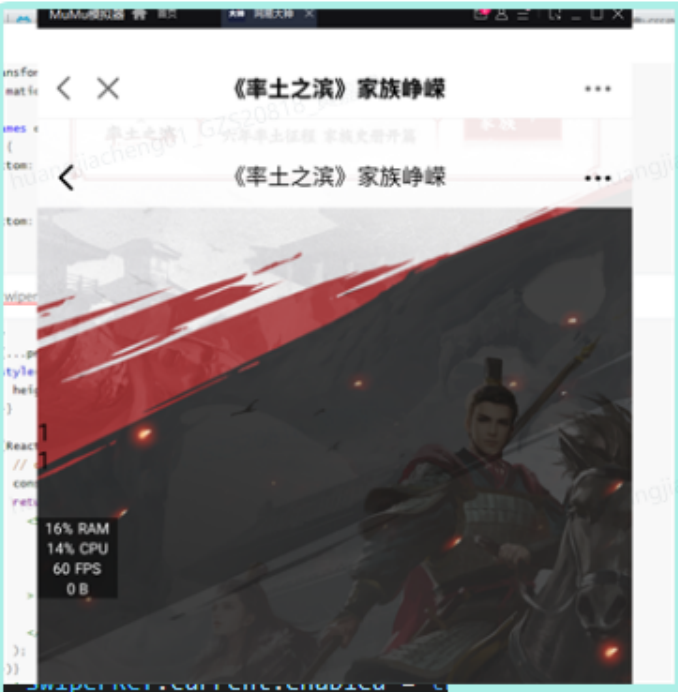
开发遇到的问题

1. 交互跳转逻辑不够明确，指的是结合前端跳转路由的设计。

> 原本是想用单路由设计项目，开发组件跳转差不多时，突然觉得有些交互跳转逻辑用单路由实现不太对劲。这个发现点是因为ui有一个设计是，首页和其他页的导航栏样式不一致，在页面显示状态更改的时候，也一同变更。这时候源于对导航栏设计的认知不足，觉得是要通过多路由跳转页面才能实现导航栏的跳转变化。于是把项目的路由设计重构成多页面路由。在此期间又发现本项目看似简单，跳转逻辑初期是觉得很混乱的。比如有些能来回跳的就需要用history,.replace 从首页跳过去就用history.push 这些都是摸着石头过河后敲定的

2. ui设计不一致和背景风格的问题

> 一个是上面说的导航栏字体颜色变化的问题，这个耗时比较久。当时做的记录。最后还是请教了丽虹之后才用了一个简单思路实现
>
> ```
> window.document.getElementsByTagName('body')[0].classList.remove('withe');
>     window.document.getElementsByTagName('body')[0].classList.add('black');
> body.withe {
>   --ds-nav-bar-title-font-color: #fff;
>   --ds-nav-bar-icon-color: #fff;
> }
>
> body.black {
>   --ds-nav-bar-title-font-color: #000;
>   --ds-nav-bar-icon-color: #000;
> }
> ```




```
if (ds.isGodlike) {
  setFullScreen(false);
  // setNavBar(false);
  window.navBar?.hide();
}
```

第一句控制上面那条有无，第二句隐藏下面那条

```
export async function setFullScreen(isOpen: boolean) {
  await ds.ready();
  ds.callHandler('setWebviewFullScreen', {
    isFullScreen: isOpen,
  });

  ds.callHandler('setStatusBar', {
    color: 'white',
  });
}
```

不过还是感觉挺奇怪了，一定要这么做才行

下午 11:52:13

底下这个ds.callHandler('setStatusBar', {
color: 'white',
});

下午 11:52:54

用了就没办法取消，对应的是图上第二条导航栏，只能用hide
其实是对这个库不熟，现在知道这俩个jdk分别是干嘛的

下午 11:54:28

然后最大问题是数据页ui和交互和运营那边不明确的问题。这就导致了后面说的为什么要进行ui大规模重构的问题。
口号是大标题，一开始是用字体去实现，没想到ui说有一些设计上的调整，后面要改成标题切图。此时武将页口号也是标题，因为一开始说是动态的，包括武将图也是动态的。到这里就发现，第一版的实现和最后的实现完全相反了。重构就是要把所有页重构成口号武将切图。最后的动态是后面运营和uil临时决定，把动态的武将口号和武将图，以人工切图的方式批量弄好和武将名映射，放到配置后台，期间一直等待所以没有先进行重构，因为也不知道运营会把图弄成什么样子。期间包括开发数据页和这个的配置数据结构设计也耗费了时间。

> 另一个就是适配问题，这次的适配说实话更多的还是ui风格设计的问题占比较大，因为她的背景图的相对参照性太明显。意思是，某些元素位置上下移很容易就造成设计的缺陷，比如重叠遮挡。小屏机屏幕太短就容易有按钮或者头部栏和文案的重叠遮挡，关键是文案是切图里面的，不在页面布局元素内。

3. swiper使用的一些问题

> 主要是2个地方使用了swiper，所以个人页最后一页不能滚动好做，虽然也是变更了几种实现方法最后敲定的。家族页swiper最后一页是家族列表，是要直接定格的。期间因为使用事件监听到滑动到最后一页时禁止下滑来实现`resistanceRatio={0}`原本这个就能解决的功能，导致期间产生一些滚动体验上的bug。

4. 虚拟列表+滚动加载的实现

> 这个只能说是误打误撞实现了。一开始还去看虚拟列表的原理，本想手动实现，后面经提示使用ahooks里面的useVirtualList。期间做了不少尝试最终实现。

```
const data = useRecoilValue(familyListState);
// arrayToGroupTwoArray 这个函数是用来实现一行展示两个item，挺麻烦的
const originalList = useMemo(() => arrayToGroupTwoArray(data), [data]);
const containerRef = useRef(null);
const wrapperRef = useRef(null);
const [list] = useVirtualList(originalList, {
    containerTarget: containerRef,
    wrapperTarget: wrapperRef,
    // 适配问题，虽然一行代码解决，但一开始因为这个质疑了本流程实现的正确性，差点推翻重做
    itemHeight: (document.documentElement.clientWidth / 750) * 100 * 3.84,
    overscan: 20,
  });
// 数据接口 滚动加载配合实现 新数据拼接到原列表
const getFamilyList = async () => {
    // if (!window.hasFamilyList) {
    const arrCopy = [...data];
    try {
      // window.hasFamilyList = true;
      res.current = await fetchFamilyList();
    } catch (e) {
      // window.hasFamilyList = false;
      console.log('e', e);
    } finally {
      // setFamilyList(res.current?.data!);
      const fetchArr = res.current?.data;
      if (fetchArr && fetchArr.length > 0) {
        // console.log('res.current', ...fetchArr);
        arrCopy.push(...fetchArr);
        setFamilyList(arrCopy!);
        setTimeout(() => {
          window.isRefresh = false;
        }, 0);
      }
    }
  };
<div className="family-list__items" ref={containerRef}>
            <div
              ref={wrapperRef}
              style={{
                margin: '0 auto',
                width: '100%',
```

```
              }}
            >
              {list.map((el) => {
                    // 原理是 检测当前渲染的虚拟列表的最后一个的index和整体列表的最后
一个index是否一致
                    // 则判定到达队尾，请求新数据进行拼接
                if (el.index === originalList.length - 1 &&
!window.isRefresh) {
                  window.isRefresh = true;
                  getFamilyList();
                }
                return (
                  <div className="family-list__row2">
                    {el.data.map((item) => (
                      <FamilyListItem data={item} />
                    ))}
                  </div>
                );
              })}
            </div>
          </div>

// 我怀疑这个把1*n数组转成2* n/2数据 的 数据结构转化功能是有库的 不过那时候自己实现了，
// 因为一些边界值也耗费了不少时间测试
export const arrayToGroupTwoArray = (alist: Array<DsType.FamilyListData>) =>
{
  const newArr: DsType.FamilyListData[][] = [];
  const len = alist.length;
  alist.forEach((el, i) => {
    if (i % 2 === 0 && i < len - 1) {
      const temArr = [];
      temArr.push(alist[i]);
      temArr.push(alist[i + 1]);
      newArr.push(temArr);
    }
    if (i === len - 1 && len % 2 !== 0) newArr.push([el]);
  });
  console.log('newArr', newArr);
  return newArr;
};
```

5. 一些内部封装的库使用不熟练 走了不少弯路
6. 字体繁体问题 搞了很多帮她一起找字体库 还处理一些字体过大按需压缩的问题