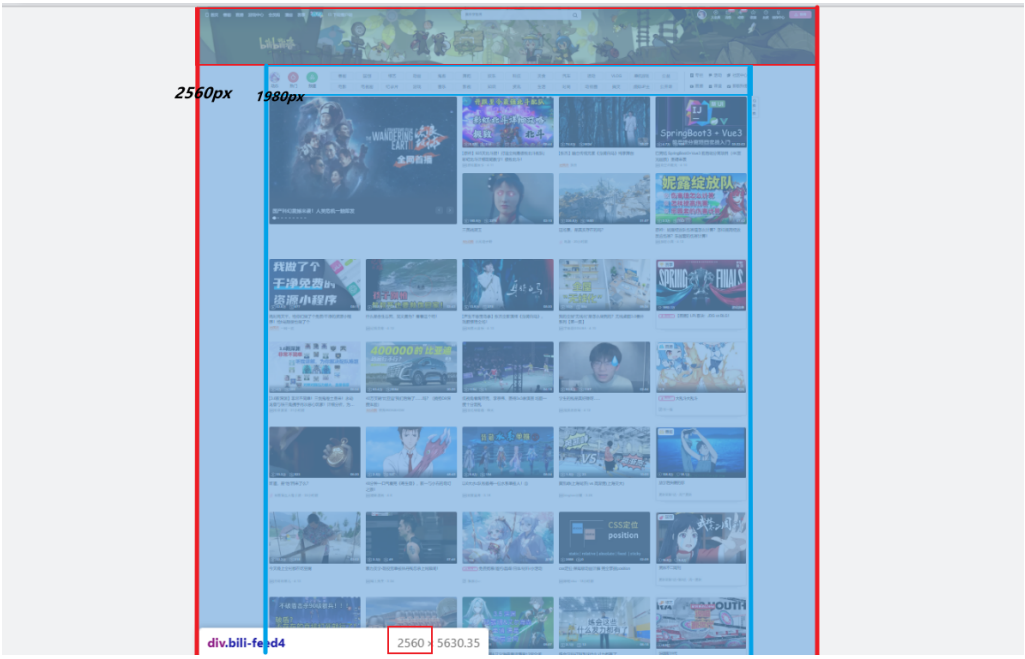
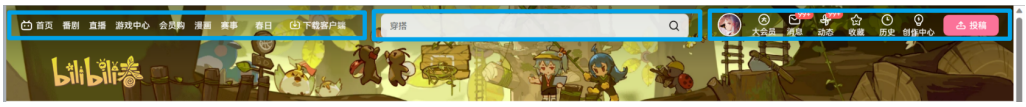
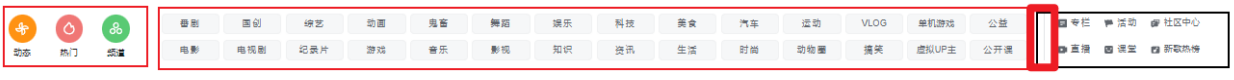
首页

页面布局

> 使用**aspect-ratio固定盒子的宽高**







瀑布流无限加载效果

监听事件触发加载

> 加载创建新盒子，并渲染视频

滑动触发

> 监听scroll 当页面滑到下面的时候就创建新的视频然后再实现懒加载效果

⚠️注意节流

```
window.addEventListener("scroll", throttle(() => {
    //回调函数
    //如果页面划到了距离最下面有一定距离的地方，就渲染新的数据
    const scrollTop = document.documentElement.scrollTop || document.body.scrollTop//文档的滚动高度
    const scrollHeight = document.documentElement.scrollHeight || document.body.scrollHeight//文档的总高度
    const clientHeight = document.documentElement.clientHeight || window.innerHeight//当前视窗的高度
    //逻辑或是为了兼容性
    if (scrollTop + clientHeight >= scrollHeight - 100) {//视窗高度加滚动高度接近总高度了
          //渲染新的html
         addVideo()
    }
}, 500))
```

每一个视频的标签对应如下

```
<section><video src="">视频</video>
    <p class="videoTitle">标题</p>
    <p class="info">info</p>
</section>
```

resize触发

> checking（）

💡 注意，当页面比较长的时候我们也要判断并用视频将页面填满

```
//但这个函数必须滚动起来才能触发，如果没有滚动,屏幕比较长，但是视频也没有加载好，就有问题
//于是打开页面先判断够不够长，不够长就继续渲染
function checking() {
    const scrollTop = document.documentElement.scrollTop || document.body.scrollTop
```

```
    const scrollHeight = document.documentElement.scrollHeight || document.body.scrollHeight
    const clientHeight = document.documentElement.clientHeight || window.innerHeight
    if (scrollHeight <= clientHeight) {
        addVideo()//添加html
        const sections = document.querySelectorAll('.content .loading')
        askForVideo(sections)//请求视频
        return checking()
    }
    else return
}
checking()

window.addEventListener('resize', checking)
```

加载过程

> addVideo()创建新盒子
> askForVideo(sections)渲染视频内容

addVideo()

- 给视频列表添加视频，每个视频附加类名为.loading，执行加载动画

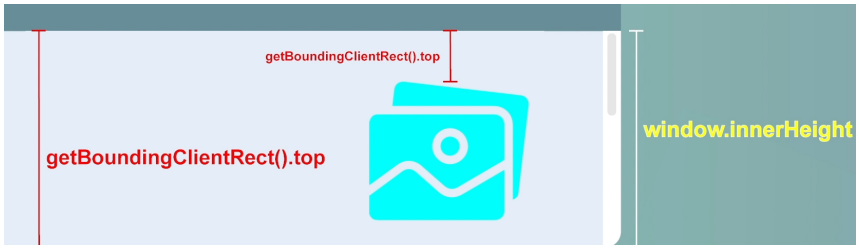
```
//添加新的视频盒子
function addVideo() {
    for (let i = 0; i < 15; i++) {
        const box = document.querySelector('.content')
        const section = document.createElement('section')
        section.classList.add('loading')
        section.innerHTML =
            `<div></div>
            <span id="a"></span>
            <span id="b"></span>
            <span id="c"></span>`
        box.appendChild(section)
    }
}
```

懒加载 async function askForVideo(sections)

- 定义一个observer
  - ajax获取当前15个视频的对象
  - .then方法，等待数据到位，渲染盒子
    - title,time,src,author,waterMask
    - 对视频执行绑定点击跳转和悬停播放的函数

- 修改section类名为.ready，取消观察

- 给sections中的每一个section添加observer观察，当section进入视窗时，执行渲染回调函数;



📸 Intersection observer 交叉观察

> 💡 **盒子类名变化**
>
> 懒加载创建盒子：loading → 用ajax获取 → reading → 渲染完成

> ⚠️ 交叉观察中：`let observer = new IntersectionObserver (callback,option)`
> 其中`option:{threshold:1}`，section全部可见时才会执行callback，与事实不符

> ⚠️ 点击跳转的url在这里被定义，通过urlParam传参，在这里我直接传入了：`replaceUrl(entry.target.firstElementChild, 'video.html?' + "src=" + `${JSON.stringify(response)}` + '&current=' + `${i}`)` 在video详情界面跳转链接会显得有一些冗长。每一次的ajax请求获得的视频列表的顺序时不同的，为了点击之后在详情界面的视频列表中获得的数据是本次ajax的结果，保证视频列表的正确性，所以才出此下策。当有更好的解决方法。

鼠标悬停功能

展现videoCard

> html video API

伪代码：

首先添加一个伪元素{

绝对定位在下方，渐变颜色}，显示的是视频的
时长和播放，弹幕等

伪元素不方便控制类名和样式，所以我们可以选择用一个真dom，通过js控制行为，操作难度会低很多

男单决赛-第1场

UP 乌拉拉拉拉拉啦 · 15小时前

男单决赛-第1场

UP 乌拉拉拉拉拉啦 · 15小时前

hover之后（mouseenter）视频开始播放，而且下方的信息消失，出现一个加入待会看列表的的图标（不实现）

```
//鼠标悬停播放视频，弹幕
function mouseenterPlay(target) {//target是个media标签
    target.addEventListener('mouseenter', () => {
        target.muted = true
        target.play()
        barSending(target)
        runningBarrage.style.display = 'block'
    })
    target.addEventListener('mouseleave', () => {
        target.pause()
        runningBarrage.style.display = 'none'

    })
}
```

⚠️如何实现伪元素里面的布局

video和.video各有两个元素，分别放左右就可以变相操作样式了
至于伪元素中的内容可以用`content:attr()`通过父元素的属性名称用js控制

> 💡 Uncaught (in promise) DOMException: play() failed because the user didn't interact with the document first：浏览器阻止了没有用户交互的自动播放，除非设置muted = true不然就不会播放

点击跳转

> url传参Params

```
//给每个视频绑定点击跳转事件
function replaceUrl(target, url) {
    target.addEventListener('click', () => {
        window.open(url)
    })
}
```

```
window.location.href = 'url'//给当前的url重新分配，当然就会取代这个页面
window.location.replace("url")//直接取代
window.open('url')//会产生新的页面，相当于a标签的target="_blank" /用这个
```

播放弹幕

> barSending()选出对应的弹幕
> creatBarrage()发送弹幕

`barSending`

- 添加计时器，1s执行一次：
  - 通过对比发送时间和当前视频currentTime，决定是否发送
  - 若发送，执行creatBarrage()函数，传入内容和容器`creatBarrage(content, target)`
    - 设置弹幕动画参数
    - 交叉观察，注意要在弹幕移入之后（threshold>0）才添加观察对象，否则一开始threshold就是0，弹幕直接被消除了
    - 当弹幕盒子移出容器的时候便删除这条弹幕并解除观察

数据

> 以名字+condition为键保存在localStroage中

```
// 弹幕数量,评论数量，点赞和播放数量
function Condition(subAmount, viewAmount, barAmount) {
    this.subAmount = subAmount
    this.viewAmount = viewAmount
    this.barAmount = barAmount
}
```